UNIVERSIDADE DO EXTREMO SUL CATARINENSE – UNESC

CURSO DE CIÊNCIA DA COMPUTAÇÃO

MAICON MATIOLA DOMINGOS

PROTÓTIPO DE APLICAÇÃO PARA COMUNICAÇÃO EM TEMPO REAL *PEER-TO-PEER* ENTRE NAVEGADORES UTILIZANDO WEBRTC

CRICIÚMA

2019

MAICON MATIOLA DOMINGOS

PROTÓTIPO DE APLICAÇÃO PARA COMUNICAÇÃO EM TEMPO REAL *PEER-TO-PEER* ENTRE NAVEGADORES UTILIZANDO WEBRTC

Trabalho de Conclusão de Curso, apresentado para obtenção do grau de Bacharel no curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC.

Orientador: Prof. Esp. Gilberto Vieira da Silva

CRICIÚMA

2019

Aos meus pais e minha namorada Ayume, que foram a luz que guiaram meus passos durante esta longa caminhada.

MAICON MATIOLA DOMINGOS

PROTÓTIPO DE APLICAÇÃO PARA COMUNICAÇÃO EM TEMPO REAL *PEER-TO-PEER* ENTRE NAVEGADORES UTILIZANDO WEBRTC

Trabalho de Conclusão de Curso aprovado pela Banca Examinadora para obtenção do Grau de Bacharel, no Curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC, com Linha de Pesquisa em Desenvolvimento para Web

Criciúma, 09 de dezembro de 2019.

BANCA EXAMINADORA

Prof. Gilberto Vieira da Silva - Esp. - (UNESC) - Orientador

Prof. Gustavo Bisognin - Me. - (UNESC)

Prof. Matheus Leandro Ferreira - Esp. - (UNESC)

AGRADECIMENTOS

Primeiramente gostaria de agradecer a Deus, por ter me concedido a capacidade, estrutura, sabedoria e força de vontade necessárias para a conclusão desta importante etapa da minha vida.

A minha namorada Ayume Mazzucco, pela companhia, apoio e compreensão durante estes cinco anos de curso de graduação, os quais enfrentamos juntos. Você é a luz que brilha, na minha trilha em minha jornada, você é meus passos lentos, meu acalento na caminhada, em seus braços amanheço, é meu endereço é minha razão, você é o meu martírio é meu delírio de exaltação.

Aos meus pais Mario e Ivete, a ambos agradeço pela criação, valores e o carinho com que me trataram durante toda minha vida, especialmente minha mãe a qual fora minha companheira inseparável durante meus anos de estudo, me esperando todas as noites, para me esquentar aquele prato de minestra, para dividir comigo meu sofrimento, meu cansaço, mais também minhas conquistas e minha felicidade.

Meu professor orientador Gilberto Vieira por ter me acolhido como seu orientando, já durante o andamento do TCC, agradeço pela sua sinceridade e cordialidade, compartilhando comigo sua sabedoria e seus conhecimentos, os quais certeza vou guardar pelo resto da vida.

Enfim gostaria de agradecer a todos os amigos e colegas que contribuíram com minha jornada acadêmica, direta ou indiretamente, não apenas aos meus amigos da faculdade mais também aos meus amigos pessoais os quais me ajudaram a tornar esta caminhada menos longa e solitária.

.

“O que sabemos é uma gota; o que ignoramos é um oceano. ”

Isaac Newton

RESUMO

Atualmente o serviço de comunicação em tempo real é uma das aplicações mais importantes da internet, esse método de comunicação possibilita aos seus usuários compartilhar uns com os outros, por razões pessoais, sociais, educacionais ou de negócios, informações como texto, áudio e vídeo. Há uma grande variedade de aplicações, com diferentes soluções de comunicação, mas, somente uma pequena parcela destes usam o conceito totalmente descentralizado baseado em *Peer-to-Peer* (P2P). Como solução para este problema, foi desenvolvido um protótipo de aplicação para comunicação em tempo real P2P entre navegadores utilizando WebRTC. O mesmo tem como objetivo permitir a comunicação direta entre dois ou mais navegadores utilizando as funcionalidades do WebRTC em conjunto com um servidor de sinalização baseado em Node.js. Os resultados alcançados no final do trabalho foram satisfatórios, uma vez que se obteve um protótipo funcional com a capacidade de transmitir texto, áudio e vídeo diretamente entre navegadores utilizando o WebRTC e que pode servir como base para desenvolvimento para futuras soluções na área da comunicação em tempo real.

Palavras-chave: *Peer-to-Peer*. Comunicação em tempo Real. WebRTC.

ABSTRACT

Today's real-time communication service is one of the most important applications of the internet, this communication method enables its users to share information such as text, audio and video with each other for personal, social, educational or business reasons. There are a wide variety of applications with different communication solutions, but only a small portion of these use the fully decentralized Peer-to-Peer (P2P) concept. As a solution to this problem, a prototype application for P2P real-time communication between browsers using WebRTC was developed. It aims to enable direct communication between two or more browsers using WebRTC features in conjunction with a Node.js based signaling server. The results achieved at the end of the work were satisfactory, as a functional prototype with the ability to transmit text, audio and video directly between browsers using WebRTC was obtained and can serve as a basis for development for future real-time communication solutions.

Keywords: Peer-to-Peer. Real-time communication. WebRTC.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interfaces*
CSS	*Cascading Style Sheets*
DOM	*Document Object Model*
ECMA	*European Computer Manufacturers Association*
HTML	*Hypertext Markup Language*
HTML5	*Hypertext Markup Language 5*
HTTP	*Hypertext Transfer Protocol*
IETF	*Internet Engineering Task Force*
IP	*Internet Protocol*
LAN	*Local Area Network*
NAT	*Network Address Translation*
NPM	*Node Package Manager*
P2P	*Peer-to-Peer*
RTC	*Real-time communication*
RTCP	*RTP Control Protocol*
RTP	*Real-time Transport Protocol*
RTSP	*Real-Time Streaming Protocol*
SIP	*Session Initiation Protocol*
SVG	*Scalable Vector Graphics*
UDP	*User Datagram Protocol*
W3C	*World Wide Web Consortium*
WebRTC	*Web Real Time Comunication*
XMPP	*Extensible Messaging and Presence Protocol*

SUMÁRIO

1 INTRODUÇÃO

Atualmente o serviço de comunicação em tempo real é uma das aplicações mais importantes da internet, esse método de comunicação possibilita aos seus usuários compartilhar uns com os outros, por razões pessoais, sociais, educacionais ou de negócios, informações como texto, áudio e vídeo (BARRY; TOM, 2009, tradução nossa). Há uma grande variedade de aplicações, com diferentes soluções de comunicação, mas, somente uma pequena parcela destes usam o conceito totalmente descentralizado do *Peer-to-Peer* (P2P) (EDÄNGE, 2015, tradução nossa).

Segundo Majid et al. (2016, tradução nossa), no mercado atual existem inúmeras aplicações para comunicação em tempo real, como por exemplo: Skype, Google Hangouts, Face Time, sendo que a grande maioria dessas aplicações utilizam a arquitetura cliente/servidor, neste modelo de aplicação os usuários precisam utilizar um agente como um smartphone, estações de trabalho e outras aplicações de hardware e software, para que posteriormente o agente se conecte a um servidor central. Ainda de acordo com o autor, usar a arquitetura cliente/servidor nesse tipo aplicações pode aumentar muito o custo do sistema, devido a serviços como configuração e manutenção.

Edange (2015, tradução nossa) aponta que usar a arquitetura cliente/servidor para atender um grande número de usuários exige um hardware mais rápido e robusto dos servidores, fator que tornaria o custo ainda mais elevado. O autor também comenta que outra desvantagem é a dependência dos clientes para com o servidor, pois caso houver algum problema com o mesmo, o sistema poderia falhar afetando todos os clientes conectados a ele.

Uma rede P2P ao contrário da arquitetura cliente/servidor disponibiliza uma comunicação direta entre os usuários sem qualquer instancia centralizada (DISTERHOFT; GRAFFI, 2015, tradução nossa). Uma rede de arquitetura P2P é formada por entidades chamadas de pares, sendo que esses podem trabalhar tanto como cliente, quanto como servidor, assim, quando atuam como cliente consomem conteúdo de outros pares, e quando atuam como servidor compartilham seus recursos, como processamento e largura de banda com o objetivo de disponibilizar conteúdo para outros pares. Esta arquitetura não precisa de infraestrutura dedicada ou controle centralizado, possui capacidade de gerir e organizar ocasionais falhas de

rede e sua população altamente transitória de pares (VINEETH et al., 2017, tradução nossa).

Atuchukwu (2009, tradução nossa), afirma que, as aplicações de comunicação em tempo real com base web tem se tornado populares devido suas vantagens diante das aplicações comuns. Essas aplicações não precisam de download, atualizações, instalação ou configuração, além disso, são independentes da plataforma, uma vez que só precisam de um dispositivo que possui um navegador web e conexão com a internet ou uma rede LAN.

O WebRTC é um projeto de código aberto suportado pelo Google, Mozilla e Opera (WEBRTC, 2018). O objetivo do projeto é ampliar o modelo de navegação web, possibilitando aos navegadores comunicação direta de áudio, vídeo e dados em tempo real utilizando a arquitetura P2P (LORETO; ROMANO, 2014, tradução nossa). Ainda segundo os autores, o World Wide Web Consortium (W3C) junto com a Internet Engineering Task Force (IETF) são responsáveis por definir para o projeto WebRTC as Application Programming Interfaces (API) de JavaScript, as tags HTML5 padrão, e os protocolos de comunicação, com objetivo de possibilitar a qualquer aplicação web com WebRTC, funcionar em qualquer dispositivo, dispondo acesso seguro aos periféricos de entrada (webcams e microfones), para troca de mídia em tempo real P2P.

Com base nas informações levantadas, esta pesquisa propõe o desenvolvimento de um protótipo de aplicação para comunicação em tempo real P2P entre navegadores utilizando WebRTC, tendo em vista que este tipo de aplicação pode ser útil para vários tipos de usuários e a API WebRTC tem um grande potencial a ser explorado por desenvolvedores web.

1.1 OBJETIVO GERAL

Desenvolver um protótipo de Aplicação para comunicação em tempo real *peer-to-peer* entre navegadores utilizando a API WebRTC.

1.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos desta pesquisa consistem em:

a) Estudar a arquitetura P2P, suas aplicações vantagens e desvantagens;

b) Compreender o conceito de *Real Time Communication* (RTC), seus modos de transmissão e principais tipos de serviço;
c) Estudar o Projeto WebRTC, sua arquitetura, APIs e principais características;
d) Identificar os navegadores a serem utilizados, e que possuem suporte ao WebRTC;
e) Realizar testes com o protótipo.

1.3 JUSTIFICATIVA

Majid et al. (2016, tradução nossa) observa que no mercado atual existem inúmeras aplicações para comunicação em tempo real, porém, a grande maioria dessas aplicações utilizam a arquitetura cliente/servidor, característica que pode aumentar muito o custo do sistema, devido aos serviços como configuração e manutenção.

Para Edange (2015, tradução nossa), outro fator determinante nos custos desse tipo de aplicação seria a necessidade de um hardware mais rápido e robusto para que os servidores possam suportar grande número de usuários. Ainda segundo o autor, outra desvantagem é a dependência dos clientes para com o servidor, pois caso houver falha no servidor central, todos os clientes conectados ao mesmo serão afetados.

Uma rede P2P ao contrário da arquitetura cliente/servidor disponibiliza uma comunicação direta entre os usuários sem qualquer instancia centralizada (DISTERHOFT; GRAFFI, 2015, tradução nossa). Vineeth et al. (2017, tradução nossa), afirma que, esta arquitetura não precisa de infraestrutura dedicada e possui capacidade de gerir e organizar ocasionais falhas de rede e sua população altamente transitória de pares.

De acordo com Atuchukwu (2009, tradução nossa), as aplicações de comunicação em tempo real possuem vantagens diante das aplicações comuns, essas aplicações não precisam de download, atualizações, instalação ou configuração, além disso são independentes da plataforma devido ao fato de serem acessadas por meio de navegadores web.

O WebRTC é um projeto de código aberto suportado pelo Google, Mozilla e Opera (WEBRTC, 2018, tradução nossa). O objetivo do projeto é ampliar o modelo de navegação web, possibilitando aos navegadores comunicação direta de áudio, vídeo e dados em tempo real utilizando a arquitetura P2P (LORETO; ROMANO, 2014, tradução nossa).

Com base nos fatores descritos anteriormente e considerando a necessidade de se implementar um protótipo de aplicação para comunicação em tempo real P2P entre navegadores, a API WebRTC mostrou ser uma alternativa bastante interessante, uma vez que é um projeto aberto e gratuito que permite desenvolver aplicações para comunicação direta em tempo real de texto, áudio, vídeo para navegadores, além disso, a iniciativa WebRTC é um projeto apoiado pelo Google, Mozilla e Opera, possibilitando que em futuro próximo o WebRTC possa ganhar ainda mais força.

1.4 ESTRUTURA DO TRABALHO

A estrutura deste trabalho de pesquisa está dividida em oito capítulos, sendo que o primeiro capítulo é constituído pela introdução ao tema da pesquisa, objetivo geral e específicos e pôr fim a justificativa do trabalho em questão.

O segundo capítulo aborda a arquitetura P2P, partindo desde os conceitos da arquitetura, suas principais características, comparativo entre a arquitetura cliente/servidor e P2P, e as aplicações da arquitetura mesma.

O terceiro capítulo trata sobre RTC ou comunicação em tempo real, descrevendo os conceitos de RTC, protocolos de transmissão e aplicações do RTC.

O quarto capítulo aborda as tecnologias para desenvolvimento web, começando com uma breve introdução à navegadores web, e a seguir abordando tecnologias como, HTML5, JavaScript, Node.js e Socket.IO.

O quinto capitulo é direcionado especialmente ao WebRTC, descrevendo as origens do projeto WebRTC, sua arquitetura, principais características e principais APIs.

O sexto capítulo aborda os trabalhos correlatos, tendo em vista que os mesmos se assemelham de alguma maneira ao trabalho desenvolvido, possuído

objetivos ou tecnologias semelhantes, servindo como importante fonte para consultas e comparações.

O sétimo capítulo aborda o processo de desenvolvimento do trabalho desenvolvido, descrevendo as etapas de metodologia, definição de ferramentas e recursos, analise de requisitos, prototipagem de telas, arquitetura do protótipo, desenvolvimento e resultados obtidos.

Por fim, o capítulo oito traz as conclusões obtidas com o desenvolvimento do trabalho, além de sugestões para trabalhos futuros.

2 ARQUITETURA *PEER-TO-PEER*

Para Detsch, Gaspary e Barcellos (2006), ainda que o compartilhamento de arquivos seja a utilização mais tradicional, os usuários domésticos já não são mais os únicos a desfrutar da estrutura *peer-to-peer* (P2P), sendo que os ambientes acadêmicos e corporativos já começam a desfrutar desta tecnologia.

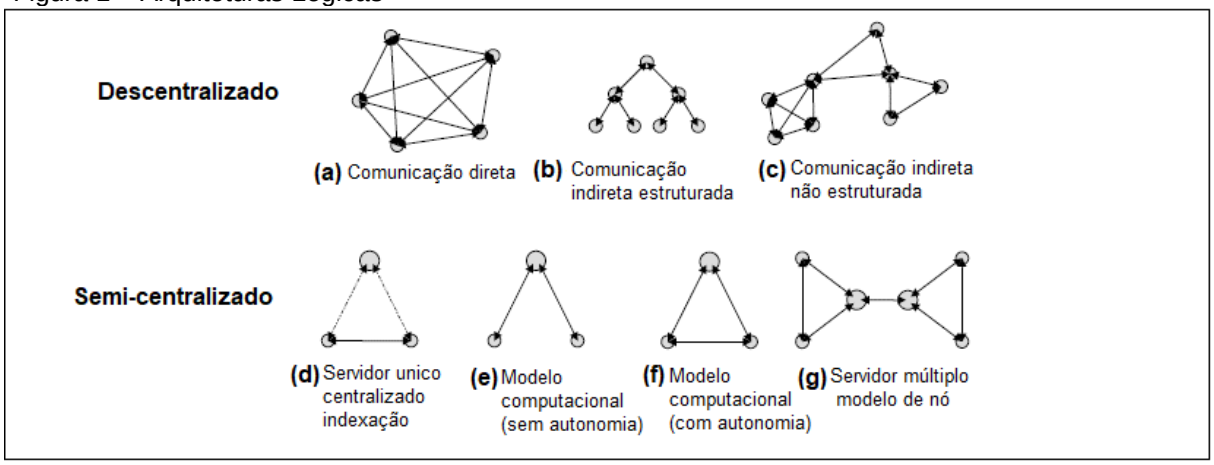
Para Zeng (2011, tradução nossa), redes P2P permitem o compartilhamento de recursos computacionais como memória, armazenamento e largura de banda. Atualmente P2P é utilizado em vários tipos de serviço como *streaming*, compartilhamento de arquivos e mensagens instantâneas.

2.1 CONCEITO DE *PEER-TO-PEER* (P2P)

Uma rede de arquitetura P2P é formada por entidades chamadas de pares, sendo que esses podem trabalhar tanto como cliente, quanto como servidor, assim, quando atuam como cliente consomem conteúdo de outros pares, e quando atuam como servidor compartilham seus recursos, como processamento e largura de banda com o objetivo de disponibilizar conteúdo para outros pares. Esta arquitetura não precisa de infraestrutura dedicada ou controle centralizado, possui capacidade de gerir e organizar ocasionais falhas de rede e sua população altamente transitória de pares (VINEETH et al., 2017, tradução nossa).

Ao estudar sistemas ou redes P2P mais detalhadamente, fica claro a existência de dois principais tipos de arquitetura. A arquitetura descentralizada onde todos os nós são considerados iguais e não existe um controle principal, e a arquitetura semi-centralizada onde há a existência de pelo menos um nó, que tem como finalidade exercer autoridade sobre os demais nós da rede (WALKERDINE; MELVILLE; SOMMERVILLE, 2002, tradução nossa), conforme pode se verificar na figura 1.

Figura 1 – Arquiteturas Lógicas



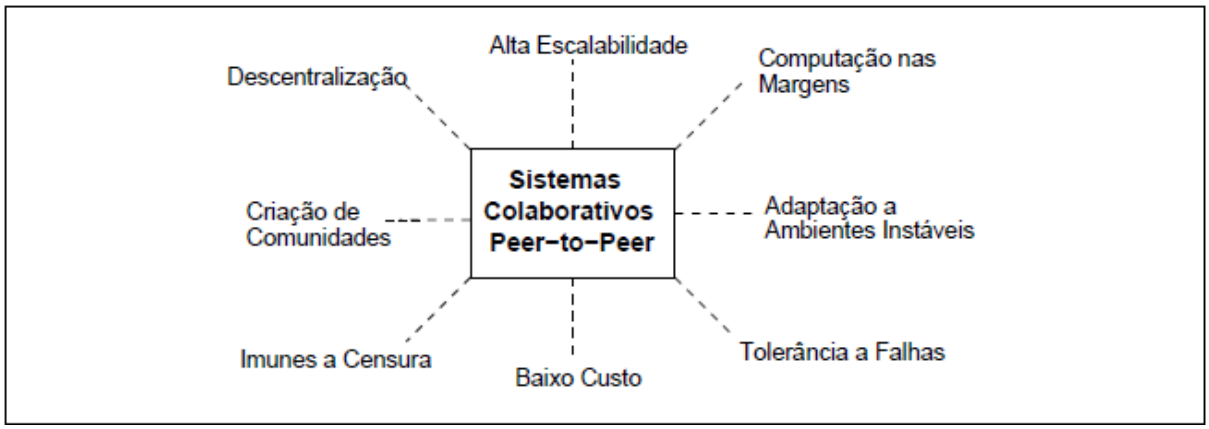
Fonte: Adaptado de Walkerdine, Melville e Sommerville (2002).

Conforme pode ser visto na figura anterior, estes são alguns dos tipos de arquitetura lógicas possíveis.

2.2 CARACTERISTICAS DA ARQUITETURA *PEER-TO-PEER*

Segundo Righi (2005), para que os sistemas P2P possam cooperar e compartilhar de recursos entre seus pares, os mesmos utilizam o conjunto de características presentes na figura 2.

Figura 2 – Características P2P



Fonte: Righi (2005).

a) A descentralização consiste em não existir um elemento central que funciona como controlador de comportamento dos demais nós. Ao contrário disso, os pares confiam uns aos outros recursos, serviços,

descoberta de outros pares, e curso de ação de forma autônoma (ROUSSOPOULOS et al., 2005, tradução nossa).

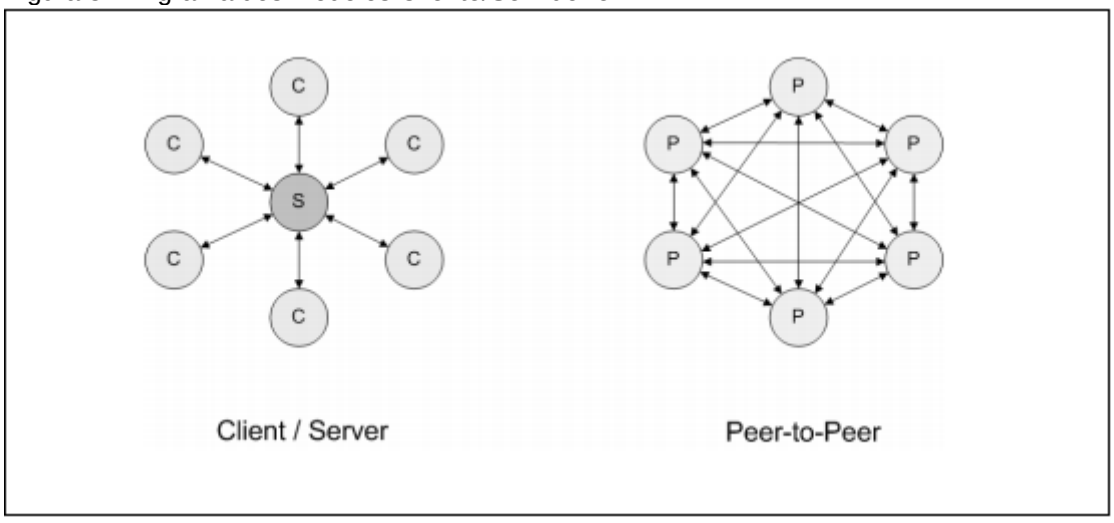
b) A escabilidade é um dos benefícios decorrentes da descentralização dos sistemas P2P, pois os pares do sistema operam como clientes e servidores simultaneamente, assim os sistemas P2P crescem em quantidade de recursos disponíveis proporcionalmente ao número de usuários, fazendo com que a escalabilidade seja no mínimo linear (KAMIENSKI et al., 2005).
c) A computação nas margens é outra característica derivada da descentralização, pois a descentralização faz com que a transmissão no ambiente P2P seja mais simétrica, ou seja, os nós que pertencem ao sistema podem tanto abrir conexões para outros nós, como também receber e tratar conexões de outros elementos do ambiente colaborativo (RIGHI, 2005).
d) Adaptação a ambientes instáveis, consistem na capacidade que sistemas P2P tem de se adequar a tipologias que utilizam recursos com *firewalls,* IPs dinâmicos, tradução de endereços de rede (NAT), a utilização do protocolo Hypertext Transfer Protocol (HTTP) para transporte de mensagens, possibilita que nós situados em redes protegidas possam ultrapassar obstáculos como *firewalls* e estabelecer comunicações bidirecionais (requisição/resposta) (RIGHI, 2005).
e) Tolerância a falhas, pois poucos ou nenhum dos nós são fundamentais para o funcionamento do sistema, sendo assim, para atacar ou desligar um sistema P2P, o invasor deve separar um grande número de nós simultaneamente (RODRIGUES; DRUSCHEL, 2010, tradução nossa).
f) Baixo custo, pois os sistemas P2P distribuem os custos de processamento e armazenamento entre todos os nós, porém, como os pares tendem a ser autônomos, é importante que os custos sejam distribuídos de forma igualitária, isso não ocorre em sistemas centralizados com muitos clientes que normalmente arcam com a maioria dos custos do sistema, o que pode se tornar demasiadamente caro, (MILOJICIC, 2002, tradução nossa).

g) Os sistemas P2P tendem a ter uma grande diversidade em termos de hardware, software, conexões de rede, localização geográfica e jurisdição, além disso, não existem entidades centralizadas capazes de controlarem os nós da rede, estas características reduzem a vulnerabilidade desses sistemas em relação a censura (RODRIGUES; DRUSCHEL, 2010, tradução nossa).
h) Comunidades P2P estão se tornando cada vez mais populares na internet, uma vez que disponibilizam uma infraestrutura onde podem ser localizadas informações e serviços, mantendo o anonimato dos pares solicitantes e provedores (XIONG; LIU, 2004, tradução nossa).

2.3 CLIENTE/SERVIDOR VS. *PEER-TO-PEER*

De acordo com Kanazawa e Takami (2018, tradução nossa), os modelos de rede podem ser classificados principalmente como cliente-servidor e P2P, sendo que no primeiro modelo o servidor central é quem administra e gerencia a rede, o sistema e seus clientes, já no modelo P2P não há servidor e os nós da rede se comunicam diretamente entre si, operando como pares, como pode ser observado na figura 3.

Figura 3 – Digrama dos modelos Cliente/Servidor e P2P



Fonte: Jimenez e Cervero (2011).

Conforme Ghareeb et al. (2015, tradução nossa), tanto o modelo cliente-servidor quanto o P2P são amplamente utilizados, ambos possuem vantagens e

desvantagens entre si. Borowsky, Logan e Signorile (2006, tradução nossa) consideram que os modelos cliente-servidor e P2P são opostos quanto a características como simplicidade, resiliência a falhas e distribuição de carga. O autor ainda menciona que, o modelo cliente-servidor é simples de implementar, especificar e gerenciar, porém a existência de um servidor central pode ser um gargalo quando o servidor estiver congestionado, além de ser um ponto único de falha no sistema, já o modelo P2P é projetado para oferecer escalabilidade e tolerância a falhas, mas com a desvantagem de ser um modelo difícil de criar e gerenciar devido à grande transição de nós no sistema.

2.4. APLICAÇÕES DA ARQUITETURA *PEER-TO-PEER.*

Segundo Rodrigues e Druschel (2010, tradução nossa), as primeiras aplicações P2P a ganharem visibilidade foram principalmente as que tinham como objetivo o compartilhamento de arquivo, porém, atualmente o uso da tecnologia P2P é muito mais diversificado e inclui além do compartilhamento de arquivo, distribuição de dados, software, mídia e telefonia. A arquitetura P2P é comumente utilizada por inúmeras aplicações de diversos gêneros, e, consequentemente, alcançando um grande número de usuários e pares (KAZUNORI; NORIO, 2015, tradução nossa).

2.4.1 Mensagens Instantâneas

Kamienski et al. (2005) apontam que as aplicações de mensagem instantânea têm se tornado uma das mais populares da internet, pois ao contrário dos e-mails em que as mensagens ficam armazenadas em um servidor para posteriormente serem entregues ao seu destinatário, as aplicações de mensagens instantâneas dispõem da entrega imediata de mensagens ao usuário, e, para que não haja incerteza na entrega, esse tipo de sistema fornece uma lista de contatos com um mecanismo capaz de identificar o estado do mesmo.

2.4.2 Compartilhamentos de Arquivos e Distribuição de Dados

Conforme Rodrigues e Druschel (2010, tradução nossa), o compartilhamento de arquivos e distribuição de dados (eDonkey e BitTorrent) são o

tipo de aplicações P2P mais populares atualmente, ambos podem ser considerados sucessores do Napster, porém, ao contrário do compartilhamento de arquivos, as redes de distribuição de conteúdo P2P não possuem componente de pesquisa de arquivos e quando seus usuários estão baixando conteúdos distintos, são desconhecidos uns aos outros, por constituírem redes separadas.

2.4.3 Telefonia

Para Rodrigues e Druschel (2010, tradução nossa), outra aplicação importante da tecnologia P2P, é o uso desta para aplicações de chamadas de áudio e vídeo, com destaque para o Skype. O Skype utiliza recursos dos nós para prover um sistema audiovisual, com conectividade independentemente da localização atual ou tipo de conexão de internet, mas, para isso ser possível, o mesmo trabalha para sanar problemas com firewalls e tradução de endereços de rede, sem estrutura centralizada, para auxiliar pares que não possuem endereços de IP públicos a conseguirem conexões.

2.4.4 Streaming de Mídia

Conforme Buford, Yu e Lua (2009, tradução nossa), o sucesso do compartilhamento de arquivos e da telefonia P2P, incentivaram o uso do P2P para aplicações de streaming de vídeo, porém, diferente das aplicações de compartilhamento de arquivos P2P, que baixa completamente o arquivo de mídia no computador do usuário, para posteriormente reproduzi-lo localmente, as aplicações de streaming de vídeo devem disponibilizar uma taxa de transferência em tempo real, igual ou superior a taxa de reprodução do vídeo para cada um dos pares do sistema, assim, se porventura houver uma demanda de streaming de 1,5 Mbps no sistema e caso exista um único par servindo como fonte de mídia, o mesmo deve fornecer uma taxa média transferência dos mesmos 1,5 Mbps. Ainda segundo os autores na eventualidade de alguma variação na taxa de transferência o sistema deve dispor de buffer que contenha um número de quadros de vídeo suficiente para evitar travamentos do lado do cliente.

3 COMUNICAÇÃO EM TEMPO REAL (RTC).

A Comunicação em Tempo Real, do inglês *Real-time communication* (RTC), é uma importante parte das redes de convergência modernas. Comunicação de voz, vídeo, bate-papo e troca mensagens são bastante populares nos dias atuais. Integrar a tecnologia RTC com o conteúdo de dados e serviços existentes tem sido difícil principalmente na web (PANDEY; BEIN, 2018, tradução nossa).

3.1. CONCEITO DE COMUNICAÇÃO EM TEMPO REAL.

Singh e Passi (2014, tradução nossa) afirmam que a comunicação em tempo real consiste na troca de informações entre remetente e destinatário, através de um canal sem que haja atraso. No campo das pesquisas, a RTC determina métodos para entrega de dados, voz e vídeo a entre emitentes e destinatários a partir de demanda em tempo real (DAVIDS; ORMAZABAL; STATE, 2014, tradução nossa).

Segundo com Davids, Ormazabal e State (2014, tradução nossa), para a implementação dos serviços RTC são empregados recursos como o protocolo de iniciação de Sessão do inglês *Session Initiation Protocol* (SIP), protocolo de transferência de hipertexto (HTTP) e códigos diversas linguagens de programação incluindo JavaScript, para que se consiga a entrega dos dados de forma síncrona.

De acordo com Singh e Passi (2014, tradução nossa), os sistemas RTC podem ser classificadas em *Hard* RTC e *Soft* RTC, as aplicações RTC *Soft*, tem como característica a capacidade de tolerar algumas perdas de mensagens, esse tipo de aplicação exige menos serviços e permite que se utilize o máximo da capacidade da rede, porém não garantem qualidade absoluta dos serviços, já as aplicações RTC *Hard,* Além de não permitir essas perdas de mensagens, podem também fornecer a qualidade total dos serviços.

3.2 PROTOCOLOS DE TRANSMISSÃO.

Para Ribeiro e Costa (2005), em uma transmissão de dados em tempo real, para que haja o processamento adequado das informações enviadas torna-se necessário atender uma série de requisitos. Devido as características de pilha do protocolo TCP/IP, com o objetivo de disponibilizar uma transmissão de dados baseada no paradigma do “melhor esforço”, o mesmo não consegue disponibilizar os requisitos necessários, portanto, novos protocolos tiveram que ser desenvolvidos para atender as necessidades para o funcionamento de transmissão de dados em tempo real. Protocolos como *Real-time Transport Protocol* (RTP), *RTP Control Protocol* (RTCP) e *Real-Time Streaming Protocol* (RTSP), são tecnologias utilizadas em transmissões em tempo real

3.2.1 Protocolo RTP

De acordo com Lee, Kim e Kim (2005, tradução nossa) o protocolo RTP foi desenvolvido pela Força Tarefa de Engenharia da Internet (IETF), a arquitetura do protocolo RTP tem como objetivo otimizar a funcionalidade de transporte em tempo real do protocolo UDP. Ainda segundo os autores o RTP não é um protocolo de transporte comum, pois não dispõe de mecanismos de confiabilidade, e não tem capacidade de distinguir a conexão, mas, é comumente utilizado como parte da aplicação e não do kernel do sistema. Por outro lado, o RTP oferece as aplicações a capacidade de diferenciar vários fluxos de mídia e acompanhar grande quantidade de estatísticas referentes a qualidade da sessão. Para Kang et al. (2009, tradução nossa) as funções principais disponibilizadas pelo RTP não garantem a qualidade do serviço, então para esta tarefa o mesmo precisa do protocolo de controle em tempo real (RTCP), para monitorar o status da rede e fornecer feedback para a camada de aplicação.

3.2.2 Protocolo RTCP

O RTCP foi criado pelo grupo de trabalho do controle de sessão multimídia da IETF e tem sido muito usado em diversas aplicações. O RTCP nada mais é que um protocolo para controle de camada de aplicação que tem o poder criar, modificar

e encerrar sessões multimídia, como por exemplo as utilizadas em aplicações de mensagens instantâneas, telefonia, e de conferências pela internet, porém, não é possível a criação de uma arquitetura de multimídia completa utilizando apenas o RTCP, pois, o mesmo não é um sistema de comunicação integrado verticalmente. O RTCP normalmente é aplicado em conjunto com outros protocolos da IETF para a construção de uma arquitetura de multimídia completa, sendo que entre esses protocolos estão o *Real-time Transport Protocol* (RTP) e o *Real-Time Streaming Protocol* (RTSP) (ABUDOULIKEMU; HUANG; YE, 2010).

Conforme Sivabalakrishnan e Manjula (2008, tradução nossa), o RTCP possibilita que as aplicações se adaptem as alterações nas condições da rede de forma autônoma, disponibilizando recursos de *feedback* referentes as condições atuais da rede e qualidade dos serviços. Ainda segundo os autores estes recursos de feedback podem ser utilizados com outras finalidades, em alguns casos, administradores de rede podem utilizar as informações de pacotes RTCP, para mensurar o desempenho de suas redes, pois, como o RTCP envia estes feedbacks não somente para o remetente, mas para todos os receptores do *multicast*, torna-se possível a percepção do problema, e se o mesmo é responsabilidade de um nó local ou da rede em geral.

Para Sivabalakrishnan e Manjula (2008, tradução nossa) existe ainda outra aplicação para os recursos do RTCP, que a partir dos relatórios do emissor e receptor RTCP oferecem a base necessária da implementação para o controle de tráfego e congestionamento, assim, com as informações dos relatórios é possível medir a flutuação da rede durante um determinado período de tempo, além de identificar e tratar qualquer possibilidade de congestionamento da mesma.

3.2.3 Protocolo RTSP

De acordo com Lee, Kim e Kim (2005, tradução nossa), o RTSP é um protocolo que tem por finalidade realizar o controle de entrega de dados em tempo real, sendo que ele atua em nível de aplicação e para que seja possível realizar entregas de dados como áudio e vídeo sob demanda e em tempo real, o RTSP fornece uma estrutura extensível e suas fontes de dados podem incluir alimentação de dados ao vivo e clipes armazenados.

Kang et al. (2009, tradução nossa) enfatizam que o RTSP é responsável por comandar a entrega de conteúdos além de disponibilizar de um variado conjunto de controles para reprodução interativa. Para o streaming auxiliado por proxy, as mensagens de controle devem ser retransmitidas pelo servidor de proxy, entre o servidor de vídeo central e os clientes.

Conforme Jianbing e Shuhui (2019, tradução nossa), o protocolo RSTP esta arquitetonicamente localizado acima dos protocolos RTP e RTCP, sendo que este protocolo tem o poder de controlar várias conexões de transmissão, além de disponibilizar formas para a escolha de canais e métodos de transmissão baseados em RTP. Resumidamente o RSTP funciona como um controle remoto de rede para servidores de dados multimídia.

3.3 APLICAÇÕES DAS COMUNICAÇÕES EM TEMPO REAL

Segundo Ribeiro e Costa (2005), várias aplicações de comunicação em tempo real tiveram o seu desenvolvimento possível, graças as tecnologias de comunicação baseadas em IP, sendo que estas aplicações de comunicação em tempo real fornecem auxilio para diversas áreas do conhecimento humano, além de serem de extrema importância no desenvolvimento e inovação do processo comunicativo, contribuindo para as comunicações entre usuários e seus serviços.

Há diversas áreas beneficiadas com as soluções em RTC, na área empresarial, por exemplo, alguns recursos como áudio conferências e videoconferências tem benefícios como agilidade e redução de custos para reuniões, na área de telecomunicações, a telefonia IP permite redução de custos com ligações de longa distância, na área da educação, as tecnologias RTC viabilizam cursos a distância, maiores recursos didáticos e mais capacidade interativa, já na área da telemedicina torna-se possível a realização de consultas e cirurgias de forma remota (RIBEIRO; COSTA, 2005).

4 TECNOLOGIAS WEB PARA DESENVOLVIMENTO

São vários os avanços recentes na chamada plataforma aberta Web, em resultado disso há também uma grande variedade de recursos usados no desenvolvimento de aplicativos nessa plataforma, entre estes recursos estão incluídos o *HyperText Markup Language* (HTML), o *Cascading Style Sheets* (CSS), a convenção *Document Object Model* (DOM), além de JavaScript e várias outras APIs de script (LORETO; ROMANO, 2012, tradução nossa). Tendo em vista a necessidade de conhecimento desses recursos para fins de desenvolvimento do protótipo proposto, estes recursos de desenvolvimento web serão verificados nos subtítulos seguir.

4.1. NAVEGADORES WEB

Os navegadores web evoluíram muito nos últimos anos, e podem ser executados em vários tipos de hardware, de telefones celulares e tablets até a computadores comuns, e há possibilidade de que os navegadores web sejam as aplicações de software mais utilizadas da história (GROSSKURTH; GODFREY, 2005, tradução nossa).

Zhu, Miao e Cai (2009, tradução nossa), afirmam que os recursos básicos dos navegadores, dispõem de um conjunto adequado de funções para navegação na internet, atendendo um grande número de usuários, uma vez que a maioria das pessoas utilizam navegadores Web apenas para visualizar páginas e navegar por links.

Serrhini (2013, tradução nossa), enfatiza que existem cinco principais navegadores contemporâneos, são eles Google Chrome, Safari, Mozilla Firefox, Internet Explorer, e Opera, sendo que estes entre outros navegadores possibilitam

acessar material na web, como textos, imagens, áudio, vídeo e animações, para que o acesso a estas mídias seja possível, o navegador por meio do protocolo de transporte HTTP se comunica com servidor web, que hospeda as aplicações. Este protocolo de transporte é responsável por definir formatos de dados, e algoritmos para empacotamento e desempacotamento dos mesmos. Estes dados podem ser HTML, CSS e outras mídias que o navegador possa interpretar com auxílio de plug-ins. Plug-ins são qualquer tipo de software implantado, que tem como finalidade ampliar as funcionalidades do navegador, um exemplo é o JavaScript que torna os sites mais interativos.

4.2 HTML5

O HTML pode ser definido como uma linguagem de programação para escrever páginas web, já o HTML5 nada mais é que a versão mais recente do HTML, que começou a ser processado a partir de 2009 e tornou-se padrão da Web em outubro de 2014 (YOON; JUNG; KIM, 2015, tradução nossa).

Segundo Stribny e Smutny (2013, tradução nossa), o objetivo do HTML5 é otimizar a linguagem disponibilizando suporte para os recursos de multimídia mais atuais e simultaneamente mantê-las mais fáceis e legíveis para os programadores e com intepretação bastante consistente para computadores e dispositivos. Os autores ainda comentam que o HTML5 traz muitos recursos novos como a integração de gráficos vetoriais escalonáveis do inglês *Scalable Vector Graphics* (SVG) e MathML para fórmulas matemáticas, além do elemento Canvas que permite a renderização dinâmica e com script de formas 2D e imagens bitmap.

Desde que o padrão HTML5 foi definido, foi amplamente apoiado por empresas e desenvolvedores, devido a isso a maioria dos navegadores atuais já suportam a tecnologia HTML5 (DAI et al., 2017, tradução nossa). Para Chen e Liu (2012, tradução nossa), o HTML5 foi projetado para ser compatível com diversas plataformas, não sendo executados apenas na plataforma do Windows, mais também em sistemas como Mac OS X e Linux, além de ser possível sua execução em dispositivos moveis como iPhone e Android, por este motivo, até setembro de 2011 cerca de 34% dos sites já tinham iniciado a utilização do HTML5.

De acordo com Nurminen et al. (2013, tradução nossa), o interesse pelo HTML5 tem aumentado fortemente, pois além de novas possibilidades de visualização multimídia, o mesmo traz uma série de possibilidades para a criação de aplicações web cada vez mais sofisticadas, e recursos inovadores como processamento paralelo, armazenamento de dados e comunicação em tempo real. Ainda segundo os autores, o HTML5 pode ser utilizado em aplicações simples, mas os grandes desafios para a linguagem estão na aplicação com alta necessidade de computação e comunicação, e implementar o streaming P2P com esta tecnologia é ainda mais desafiador.

4.3 JAVASCRIPT

O JavaScript é uma linguagem de script criada pela Netscape para o desenvolvimento de páginas HTML interativas, este por sua vez respeita os padrões de conformidade ECMAScript. Normalmente os códigos da linguagem JavaScript estão permeados no código HTML, e no momento em que o navegador realiza o download da página na web, este fica responsável por analisar, compilar e executar o script (HALLARAKER; VIGNA, 2005, tradução nossa).

Para Jibaja *et al.* (2015, tradução nossa), cada vez mais a computação é realizada em navegadores web, e como o JavaScript é a linguagem principal neste meio, o número de aplicações complexas desenvolvidas em JavaScript tem aumentado significativamente. Outro motivo pelo qual o JavaScript está se tornado popular rapidamente, é o fato de possibilitar aos desenvolvedores a criação de aplicações avançadas, com interface de usuários, vastos recursos e com portabilidade para as plataformas desktop e de dispositivos móveis (SRIDHARAN *et al.*, 2012, tradução nossa).

Park e Ryu (2015, tradução nossa), evidenciam que devido a sua popularidade, o JavaScript ampliou suas áreas de aplicação, não se limitando mais a implementação de scripts simples, sendo que atualmente, o mesmo é utilizado por desenvolvedores web para a construção de aplicações de grande escala, como jogos, compiladores e até mesmo sistemas operacionais, por isso, analisar estaticamente as aplicações JavaScript é bastante desafiador.

O JavaScript é definido como uma linguagem de tipificação dinâmica, podendo de adaptar aos paradigmas de programação procedural, orientada objeto, funcionais e prototípicos, sendo que todas estas características com certeza tornam o

JavaScript uma linguagem muito poderosa, porém, as aplicações JavaScript possuem desvantagens como, serem difíceis de implementar, depurar e manter (ESHKEVARI et al., 2017, tradução nossa).

4.4. NODE.JS

O Node.js pode ser definido como uma plataforma de software voltada para as aplicações de rede, que tem como objetivo simplificar a implementação de serviços de rede, incluindo aplicações web, tornando-os mais rápidos e escaláveis. O projeto do Node.js foi lançado em 2009 e desde então o mesmo ganhou bastante relevância entre as comunidades de desenvolvedores, além disso, grandes empresas do setor web como por exemplo Yahoo, tem contribuído para o desenvolvimento da plataforma, já outras como a Microsoft integraram em sua plataforma na nuvem o suporte ao Node.js (OJAMAA; DÜÜNA, 2012, tradução nossa).

Para Ojamaa e Duuna (2012), apesar do projeto Node.js possa ser considerado recente, pontos positivos como, sua abordagem ao desenvolvimento para aplicações de rede, escalabilidade, e reutilização de tecnologias mais populares, transformaram o Node.js em uma alternativa interessante para as plataformas mais tradicionais.

Segundo Tilkov e Vinoski (2010, tradução nossa), o Node.js é um ambiente JavaScript no lado do servidor, onde o mesmo é baseado na implementação em tempo de execução do Google o chamado “V8”, sendo que tanto o V8 quanto o Node.js são comumente implementados em C e C++, e ambos têm como princípio o foco no desempenho e baixo consumo de memória, porém, o V8 foi projetado para suportar o JavaScript no navegador, e já o Node.js tem como objetivo suportar processos de servidor de longa duração.

O Node.js diferente da maior parte dos ambientes atuais, não utiliza vários threads para suportar a execução simultânea da lógica de negócios, ao invés disso é baseado em um modelo de eventos assíncrono de entrada e saída, tornando o JavaScript parte importante para essa abordagem, uma vez que o mesmo suporta retornos de chamadas de eventos e sua natureza funcional torna extremamente fácil criar objetos de funções que podem ser utilizados para esses fins (TILKOV; VINOSKI, 2010, tradução nossa).

Chitra e Satapathy (2017, tradução nossa) afirmam que a arquitetura do Node.js baseada em eventos, trabalha executando um único thread, não existe sincronia em nível de aplicativo, uma vez que o thread for executado em um único encadeamento, o que significa que todas solicitações que chegam são executadas pelo próprio encadeamento. Ainda conforme os autores, esse tipo de abordagem torna o desenvolvimento da aplicação mais simples, e uma vez que não é possível que os aplicativos manipulem outras solicitações, fazendo com que cálculos complexos e o bloqueio de chamadas de entrada e saída sejam executados pelo encadeamento de eventos, sendo que esse tipo de processo possibilita que o Node.js utilize recursos do sistema de forma mais eficiente quando comparado a plataformas com arquiteturas multi-thread.

De acordo com Tilkov e Vinoski (2010, tradução nossa), a comunidade de desenvolvedores cunhou o que pode ser chamado de um “ecossistema” de bibliotecas para ser compatível com o Node.js. O gerenciador de pacotes do Node.js, o “NPM” possibilita a instalação de bibliotecas e suas dependências, além de permitir que muitas outras bibliotecas desenvolvidas para serem utilizadas pelo JavaScript do lado do cliente, possam ser utilizadas junto com o Node.js.

4.5 SOCKET.IO

O Socket.IO pode ser definido como uma bibliotecaque tem como funcionalidade, permitir a comunicação em tempo real, bidirecional e baseada em eventos entre o navegador e o servidor. E para que isso aconteça o Socket.IO necessita apenas de um servidor Node.js e da biblioteca JavaScript que é executada pelo navegador, pelo cliente (SOCKET.IO, 2019, tradução nossa).

Ainda segundo o autor, embora o Socket.IO utilize WebSocket como transporte sempre que possível, o mesmo acaba acrescentando metadados a cada pacote, tornando a comunicação entre Socket.IO e WebSocket incompatível. Os metadados adicionados consistem principalmente em: tipo de pacote, espaço para o nome e id de confirmação da mensagem.

Para Sooryavanshi, Upganlawar e Bhosle (2017, tradução nossa), o Socket.IO contribui para a velocidade de aplicações de comunicação bidirecional em tempo real, evitando os requisitos adicionais que são necessários ao protocolo *HTTP*,

além disso, outra característica do Socket.IO é o fato de o mesmo não fechar após ter uma conexão aberta, estando sempre disponível para novas conexões, diferentemente dos servidores comuns que ao abrirem uma nova conexão se fecham a todas as demais.

O autor ainda afirma que as características citadas anteriormente reduzem o esforço para a comunicação e garantem uma utilização mais eficiente da largura de banda disponível, possibilitando que o Socket.IO, transmita a comunicação em tempo real, com um atraso praticamente imperceptível.

Outra vantagem do Socket.IO é o suporte a *Rooms*, que são uma espécie de canal arbitrário, nas quais os soquetes podem entrar ou sair. A partir dessas *Rooms* é possível emitir mensagens a todos os soquetes participantes da mesma, tornando esse um recurso bastante interessante quando se deseja enviar mensagens para um determinado grupo de usuários, ou para um determinado usuário conectado aos demais (SOCKET.IO, 2019, tradução nossa).

5 WEBRTC

Nos últimos anos a comunicação por voz e vídeo entre os navegadores tem ganhado destaque nos dois principais órgãos de padronização da internet, a Força-Tarefa de Engenharia da Internet do inglês *Internet Engineering Task Force* (IETF) e o *World Wide Web Consortium* (W3C) (AMIRANTE *et al.*, 2013, tradução nossa).

O *Web Real Time Communication* (WebRTC), ocupa um papel importante no desenvolvimento para a comunicação entre navegadores web, uma vez que o mesmo possibilita a comunicação em tempo real, além de permitir comunicação baseada em UDP com o intuito de complementar a comunicação TCP comumente utilizada em conjunto com o protocolo HTTP (NURMINEN *et al.*, 2013, tradução nossa). O WebRTC possibilita que navegadores tenham capacidade para lidar recursos para comunicação áudio e vídeo sem necessidade de instalação de softwares e plug-ins adicionais (ZEIDAN; LEHMANN; TRICK, 2014, tradução nossa).

5.1 O PROJETO WEBRTC

Em maio de 2011 o Google foi responsável pela publicação de uma nova tecnologia de código aberto, com componentes poderosos para a comunicação em tempo real na web, o WebRTC. Participaram do desenvolvimento desta tecnologia,

empresas como Google, Mozilla e Opera (SREDOJEV; SAMARDZIJA; POSARAC, 2015, tradução nossa).

Porem foi a partir do esforço conjunto e sem precedentes da IETF e W3C, a comunicação em tempo real entre navegadores passou a ser possível. A IETF teve o foco do seu trabalho voltado para a arquitetura e protocolos, identificando os eventos e informações de estado que precisam estar disponíveis aos desenvolvedores, e por sua vez a W3C ficou responsável pelo desenvolvimento de APIs JavaScript em nível de aplicação, com a finalidade de permitir aos desenvolvedores mais praticidade e facilidade para a implementação de aplicações multimídia em tempo real na web (ROMANO; LORETO; DAVIDS, 2017, tradução nossa).

Ainda segundo os autores, a união entre IETF e W3C teve como resultado o WebRTC 1.0, que tem como objetivo permitir com que navegadores, provedores de telecomunicações, aplicações, e desenvolvedores web, juntem-se em um conjunto bem definido de protocolos e APIs que serão responsáveis pela disseminação da interoperabilidade.

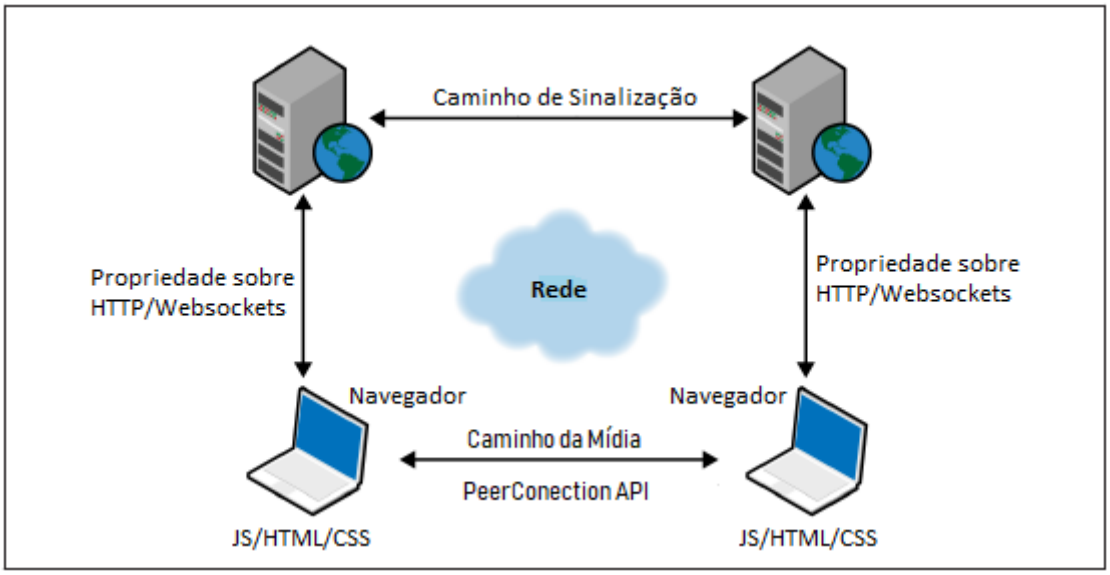
O projeto WebRTC e consiste em uma plataforma aberta e livre de royalties, que permite comunicação em tempo real de alta qualidade entre navegadores que possuam suporte ao WebRTC, são estes Google Chrome, Opera, Mozilla Firefox (AMMAR, 2018, tradução nossa).

A partir de seus padrões e capacidades, o WebRTC oferece uma nova visão e dimensão para aplicações de comunicação em tempo real, fornecendo a navegadores web recursos para transmissões de áudio, vídeo e dados (ZEIDAN; LEHMANN; TRICK, 2014, tradução nossa).

5.2 ARQUITETURA WEBRTC

Loreto e Romano (2014, tradução nossa), defendem que o WebRTC estende do modelo clássico da arquitetura web, porem introduzindo os paradigmas da comunicação P2P entre navegadores, outro fato arquitetura WebRTC é inspirada no trapézio SIP, onde ambos navegadores estão executando aplicações web que estão em servidores diferentes, como pode ser visto na figura 4.

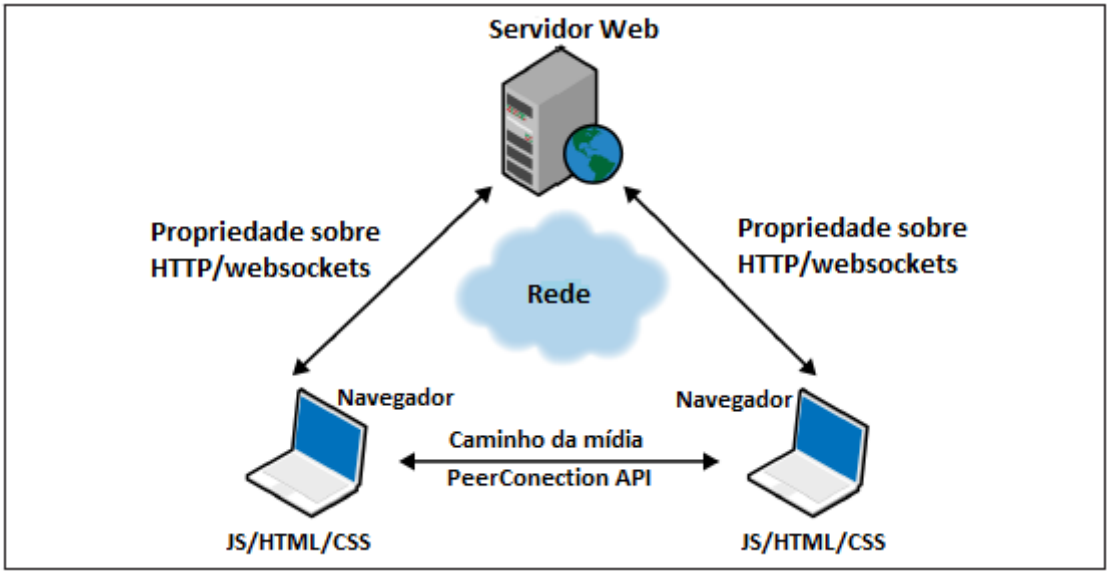
Figura 4 – Arquitetura de trapézio WebRTC



Fonte: Adaptado de Loreto e Romano (2014).

Porem de acordo com os autores o cenário mais comum para a utilização do WebRTC provavelmente é aquele em que ambos os navegadores estão acessando a mesma aplicação web, onde neste caso o trapézio visto anteriormente acaba se tornando um triangulo como mostrado na figura 5.

Figura 5 – Arquitetura de triangulo WebRTC



Fonte: Adaptado de Loreto e Romano (2014).

Segundo Pierleoni *et al.* (2016, tradução nossa), outro ponto importante das implementações WebRTC é o procedimento de sinalização, este por sua vez deve seguir a ideia geral do WebRTC, que é fazer com que os desenvolvedores de aplicações sejam os responsáveis pela implementação da sinalização, mantendo esta

funcionalidade o mais distante possível do núcleo WebRTC, uma vez que é provável que aplicativos diferentes necessitem de protocolos diferentes.

Atualmente diversos tipos de protocolos são utilizados para a sinalização WebRTC, entre os mais populares estão o protocolo SIP e XMPP (FERNANDEZ *et al.*, 2013, tradução nossa). Ainda para os autores o objetivo dos protocolos de sinalização é possibilitar a negociação dos parâmetros de transporte e formatos de mídia empregados na comunicação entre dois terminais.

5.3 PRINCIPAIS CARACTERÍSTICAS

Segundo Witherspoon *et al.* (2014, tradução nossa), o WebRTC possui algumas características que tornam o projeto bastante interessante, como ser um projeto gratuito e de código aberto, estar presente em navegadores como Chrome, Firefox e Opera, permitir que navegadores acessem periféricos, como câmera e microfone para capturar mídia e utiliza-las em transmissões em tempo real.

Para Boubendir, Bertin e Simoni (2015, tradução nossa), além das características citadas anteriormente, o WebRTC ainda possui outros predicados como, não necessitar de plug-ins ou extensões para funcionar, ter uma integração de custos reduzidos por se utilizar da infraestrutura de rede existente, e interoperabilidade, pois pode funcionar em qualquer dispositivo, uma vez que a ferramenta é integrada a navegadores web.

Para Johnston e Burnett (2014, tradução nossa), essas características e outras características do WebRTC são bastante relevantes, uma vez que não estão disponíveis em outros tipos de ferramentas para comunicação em tempo real, como pode ser observado na figura 6.

Figura 6 – Características do WebRTC.

Característica	Fornecido por	Importância
Independência de plataforma e dispositivo.	APIs padrão de W3C, protocolos padrão de IETF.	Os desenvolvedores WebRTC podem escrever códigos em HTML5 que serão executados por diferentes sistemas operacionais, navegadores e dispositivos, móvel e desktop.

Comunicação segura de voz e vídeo.	RTP Protocol (SRTP) criptografia e autenticação segura	Browsers são utilizados em diferentes ambientes e através de redes WiFi não seguras. Criptografia significa que os outros não podem ouvir ou gravar voz ou vídeo.
Qualidade avançada de voz e vídeo.	Codec Opus áudio, codec de vídeo VP8, e outros.	Por possuir codecs padrão, assegura-se a interoperabilidade e evita download de codecs, uma forma que sites maliciosos utilizam para instalar vírus e spywares. Além disso novos codecs podem adaptar-se quando o congestionamento de rede é detectado.
Estabelecimento de sessão confiável.	Infiltração por através de NAT.	Mídia direta entre os navegadores é visivelmente de maior qualidade e confiabilidade que a mídia retransmitida pelo servidor. Além disso, a carga nos servidores é reduzida.
Fluxos de mídia e múltiplos tipos de mídia enviados através de um único transporte.	Protocolo RTP e extensões *Session Description Protocol* (SDP).	Estabelecer meios diretos usando perfuração pode levar tempo. O envio de todas as mídias em uma única sessão também é mais eficiente e confiável.
Adaptação às condições da rede	Multiplexando os protocolos RTCP e *Secure Audio Video Profile with Feedback* (SAVPF)	Comentários sobre as condições da rede são essenciais para o vídeo, e são especialmente importantes para as sessões de grande largura de banda e alta definição em WebRTC.
Suporte para vários tipos de mídia e múltiplas fontes de mídia.	APIs e sinalização para negociar o tamanho / formato de cada fonte individualmente	A capacidade de negociar com cada par individualmente resulta em uso mais eficiente da largura de banda e outros recursos.
Interoperabilidade com sistemas de comunicação VoIP e vídeo que utilizam SIP, Jingle, e PSTN	Padrões *Secure* RTP (SRTP) e SDP e outras extensões.	Existem sistemas de VoIP e de vídeo que podem trabalhar com sistemas WebRTC, utilizando protocolos padrão.

Fonte: Adaptado de Johnston e Burnett (2014).

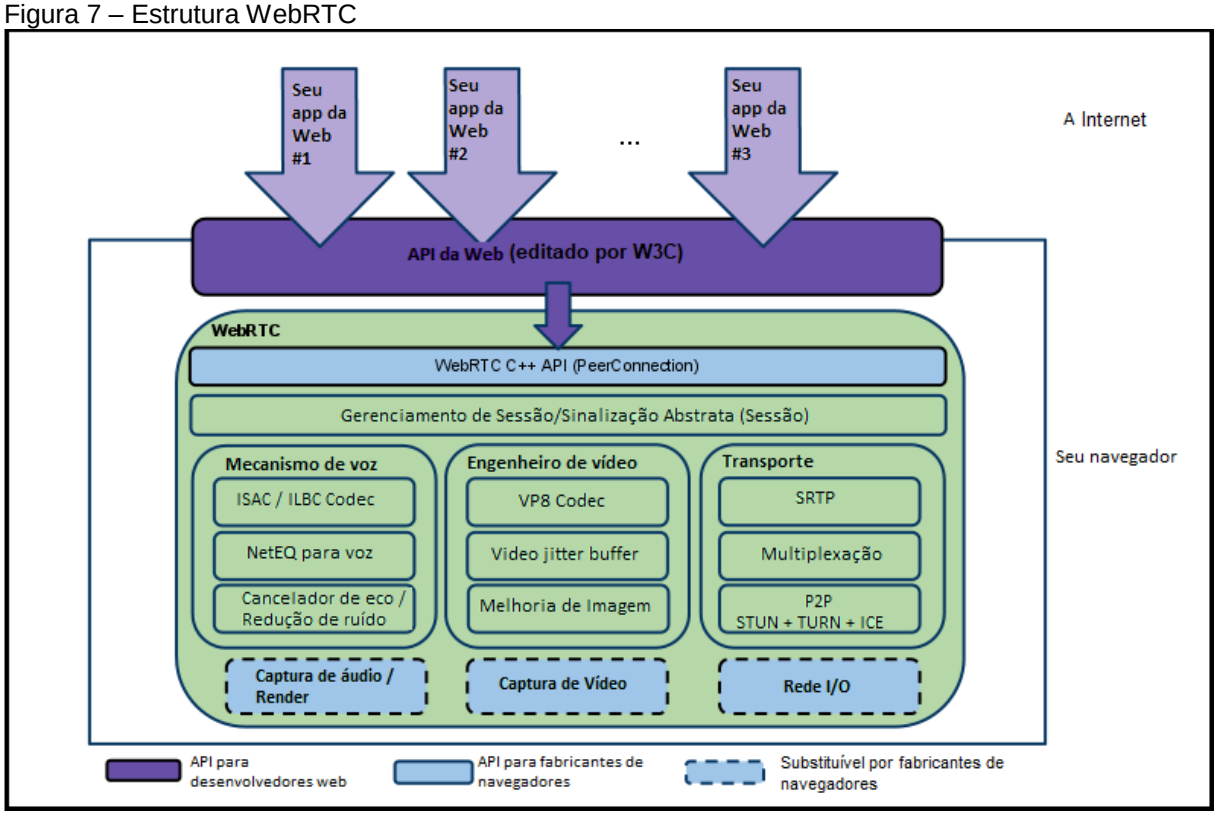
A figura anterior lista algumas das características do WebRTC, em conjunto com os componentes que as fornecem, e a importância dos mesmos para o projeto.

5.4 APIS DO WEBRTC

Segundo Sredojev, Samardzija e Posarac (2015, tradução nossa), quando se fala de WebRTC, é importante saber que não se trata de apenas uma única API mais sim de várias de APIs e protocolos que são definidos por órgãos como W3C e o IETF, e tem como objetivo prestar suporte para diferentes navegadores e sistemas operacionais.

Observando a estrutura do WebRTC, é possível identificar duas camadas distintas de APIs: A camada da API WebRTC C++ que é de interesse dos desenvolvedores de navegadores e a camada da Web API, que é voltada para desenvolvedores web (WEBRTC, 2019, tradução nossa). A figura 7 mostra a arquitetura geral do WebRTC e distinção das camadas e APIs e seus componentes.

Figura 7 – Estrutura WebRTC



Fonte: Adaptado de WebRTC (2019).

Porem de acordo com Sredojev, Samardzija e Posarac (2015, tradução nossa), as principais APIs utilizadas implementações com WebRTC são:

a) *MediaStream*: que possibilita que um navegador web possa acessar periféricos como câmera frontal e microfone;
b) *RTCPeerConnection*: Configura chamadas de áudio e vídeo;
c) *RTCDataChannel*: Permite que os navegadores web possam enviar dados por meio de conexões P2P;

5.4.1 MediaStream

A API *MediaStream* é responsável por controlar os fluxos sincronizados de mídia, este fluxo possui apenas uma entrada e uma saída, e a fonte dos dados da entrada normalmente pode se tratar de um dispositivo físico como uma câmera ou microfone, por sua vez a saída pode ser encaminhada para um ou mais destinos como um elemento de vídeo ou para API *PeerConnection* (SREDOJEV; SAMARDZIJA; POSARAC, 2015, tradução nossa).

Ainda segundo os autores, os principais componentes API *MediaStream* são eles: a interface *MediaStream* e o *MediaStreamTrack*, o primeiro representa o fluxo de mídia, podendo este fluxo ser dividido em várias partes chamadas faixas, as mesmas são de um tipo explícito como áudio e vídeo. Já *MediaStreamTrack* representa o tipo de mídia que conseguida a partir da fonte de entrada.

5.4.2 RTCPeerConnection

A API *RTCPeerConnection* tem como função permitir que os pares possam estabelecer uma conexão direta entre navegadores web, descrevendo a conexão por meio por meio do protocolo SDP (PIERLEONI *et al.*, 2016, tradução nossa).

Os autores também enfatizam que, para o início das comunicações é necessário que se tenha uma fase de sinalização por meio de um servidor, nessa fase um par envia uma oferta SPD para outro par passando pelo servidor, os pares em questão comutam e adicionam seus protocolos de estabelecimento de conectividade Interativa do inglês Interactive Connectivity Establishment (ICE), e caso a conexão seja estabelecida corretamente, os pares conectados podem enviar e receber dados a outros pares da rede através do navegador.

5.4.3 RTCDataChannel

A API *RTCDataChannel* desempenha o papel de ser o principal canal de comunicação para a troca de dados entre pares, ou seja, é o responsável pela tarefa por transmitir os dados diretamente de uma extremidade a outra da conversa, pois mesmo que existam outros canais de comunicação entre os pares, todos esses são intermediados por um servidor e não são capazes de conectarem os pares diretamente, diferente da API *RTCDataChannel* (LI *et al.*, 2019, tradução nossa).

Ainda conforme os autores, o *RTCDataChannel* possui semelhanças com o WebSocket conhecido, mais diferente deste a API do WebRTC possibilita que os navegadores formem um canal ponto a ponto de baixa latência e alta capacidade de transmissão de dados, com recursos de transporte definidos pelo usuário.

5.5 SEGURANÇA

Segundo Jaouhari (2018, tradução nossa), uma das principais preocupações em sistemas de comunicação em tempo real, é fazer com que protocolos de comunicação sejam capazes de garantir a segurança do sistema e de seus usuários, especialmente em relação a autenticação, integridade e confiabilidade.

A segurança também é um dos maiores desafios do WebRTC, considerando este fato, as organizações responsáveis pelas padronizações do projeto, estão investindo grandes esforços para solucionar possíveis problemas em relação a realização de chamadas na Web (FERNANDEZ *et al.*, 2014).

Durante o processo de desenvolvimento do WebRTC, o IETF e o W3C levantaram alguns pontos importantes em relação à segurança (JAOUHARI, 2018), especificamente:

a) Um problema comum em aplicações Web: o fato de que o JavaScript pode ser baixado de qualquer site e normalmente sem o consentimento do usuário;
b) O usuário deve ter a capacidade de controlar o acesso a seus dispositivos de entrada/saída como por exemplo câmeras e microfones;

c) Informações de cunho particular como identidade, localização e outras, não devem ser reveladas sem a aceitação do usuário, e deve se ter a disposição a possibilidade de efetuar chamadas anônimas;
d) Autenticação e confidencialidade devem estar asseguradas durante toda comunicação;
e) O usuário só deve receber o trafego de sinalização prioritário caso, não aceite a chamada recebida;

Com base nos pontos levantados, a segurança do WebRTC baseia-se em cinco pontos principais (RESCORLA, 2019, tradução nossa).

a) O aplicativo JavaScript deve ser executado dentro do sandbox do navegador;
b) A segurança está sujeita ao nível de confiança do navegador e do fornecedor dos serviços WebRTC;
c) A comunicação em geral deve ser criptografada, afim de protege-la das ameaças externas;
d) É requisito obrigatório obter o consentimento explicito dos usuários, para se ter acesso a dispositivos multimídia;
e) Requisitos como autenticação, verificação de identidade e de origem são indispensáveis;

De acordo com Jaouhari (2018, tradução nossa), a segurança no WebRTC pode ser representada por quatro partes distintas, que representam de forma geral os pontos mais relevantes a serem garantidos, para que se possa conseguir a segurança do sistema como um todo, são elas o navegador, o servidor dos serviços WebRTC, a sinalização, e a segurança das trocas P2P.

Para Desmet e Johns (2014, tradução nossa), não é surpresa, que devido à complexidade e dos muitos modelos de comunicação e aplicação encontrados no WebRTC, considerações e estudos sobre a segurança do mesmo, manterão profissionais da área ocupados pelos próximos anos.

6 TRABALHOS CORRELATOS

Durante o processo de levantamento bibliográfico e pesquisa para a execução desse trabalho, foram encontrados artigos e trabalhos que abordam temas presentes neste projeto.

6.1 *P2P MEDIA STREAMING WITH HTML5 AND WEBRTC*

Este artigo foi publicado na Conferência do IEEE 2013 sobre Workshops de Comunicação Computacional (INFOCOM WKSHPS) em Turim na Itália, tendo como autores Jukka K. Nurminen, Antony J.R. Meyn, Eetu Jalonen, Yrjo Raivio e Raul Garcıa Marrero. O artigo em questão tem como objetivo analisar se os novos padrões HTML5 junto com o WebRTC são adequados para o uso em streaming P2P, avaliando os desafios de desempenho e propondo soluções (NURMINEN et al., 2013, tradução nossa).

Em um primeiro momento os autores do artigo acabam identificando quais os principais desafios da implementação de streaming P2P com HTML5, citando inicialmente que pelo fado de padrões estarem em desenvolvimento, implementações dos navegadores acabam por apresentar problemas com funções, estabilidade e interoperabilidade, e em seguida é relatado que o streaming de vídeo com o uso da tecnologia proposta tem como desvantagem alocar uma parte considerável do desempenho da CPU, o problema torna-se ainda mais crítico em dispositivos moveis. Assim os autores do artigo propõem o desenvolvimento de um sistema experimental para o serviço de streaming P2P utilizando as novas APIs do HTML5 e WebRTC padronizadas recentemente.

O artigo termina concluindo que, o HTML5 e WebRTC podem sim, ser utilizados em uma aplicação de streaming P2P, porem com algumas a limitações principalmente em relação ao desempenho de implementações nos navegadores atuais. Todavia seriam necessários mais estudos para que seja possível determinar se a fonte dos problemas em relação ao desempenho teria origem mesmo dos navegadores ou se seriam frutos de implementações HTML5 e WebRTC prematuras.

6.2 WEBRTC - EVOLUÇÃO NA WEB

Este trabalho foi realizado por Roberto Oliveira Rocha para dissertação de mestrado em Sistemas e Tecnologias de Informação para as Organizações, pelo Instituto Politécnico de Viseu, em Viseu, Portugal, na data de outubro de 2014. Rocha (2014) defende que, o trabalho tem como objetivo apresentar a tecnologia WebRTC, explicar seu funcionamento, capacidades e funcionalidades junto com outras tecnologias que cercam o WebRTC e que permitem seu funcionamento.

O autor afirma ainda que se pretende também estudar o impacto da tecnologia pode ter no futuro do desenvolvimento Web, além de apresentar uma aplicação com o uso do WebRTC tornando mais claro as funções e possibilidades da tecnologia.

Segundo o autor durante o processo de desenvolvimento do trabalho, foi executada uma investigação fundamentada e bem estrutura, onde foram utilizados vários processos voltados ao tema e que permitiram consolidar, clarificar e criar novos conhecimentos baseando-se principalmente em seis pontos principais: Proposito, Produtos, Processos, Participantes, Paradigmas e Apresentação.

Com o desenvolvimento do trabalho conclui-se que o estudo, permitiu que se obtivesse conhecimento e compreensão sobre as necessidades, capacidades de evolução, dificuldades de implementação e funcionamento do WebRTC, além da possibilidade de que a implementação desenvolvida possa servir como base para outras soluções na área da comunicação em tempo real.

6.3 SOLUÇÕES *OPEN SOURCE* PARA INTEROPERABILIDADE ENTRE SISTEMAS DE VIDEOCONFERÊNCIA E WEBCONFERÊNCIA

Escrito por Alexandre Kreismann dos Santos, para a obtenção do grau de Bacharel em Ciência da Computação, na Universidade Federal do Rio Grande do Sul, Porto Alegre, no ano de 2017. Este trabalho tem como missão estudar os principais padrões e tecnologias na área de videoconferência, além de analisar soluções *open source* que sejam compatíveis as mesmas, possibilitando que Webconferência possam interoperar com vários sistemas de videoconferência, softwares e equipamentos (SANTOS, 2017).

O trabalho foi desenvolvido a partir do estudo dos principais e mais atuais padrões e tecnologias utilizados na implementação de sistemas de videoconferência, em seguida foram levantadas soluções open source compatíveis com os padrões estudados com a finalidade de integrar as soluções e tecnologias para que fosse alcançada a maior interoperabilidade possível e por fim aplicar os conhecimentos obtidos na realização de um estudo de caso com um sistema de Webconferência.

O autor conclui que a pesquisa efetuada durante o desenvolvimento do trabalho pode apresentar meios que possibilitem sistemas de Webconferência interoperarem com diversos softwares e equipamentos de videoconferência, e ainda com base nos estudos ficou claro a possibilidade de realizar melhorias e encontrar novas formas de interoperabilidade.

6.4 SISTEMA DE ATENDIMENTO AO CONSUMIDOR UTILIZANDO A TECNOLOGIA WEBRTC

Este trabalho foi escrito por Lucas Rinaldi, como requisito para obtenção do grau de Bacharel em Sistemas de Informação pela Universidade Federal de Santa Catarina, em Florianópolis, no ano de 2018. Para Rinaldi (2018), o trabalho tem como objetivo aprimorar os métodos como *e-commerce*, suporte técnico e serviços online fornecem atendimento ao cliente, desenvolvendo um sistema de comunicação por vídeo baseado em WebRTC, e que possa ser adicionado a sites de forma simples e rápida.

Para o desenvolvimento do trabalho, em um primeiro momento o autor fez um levantamento de requisitos, através de artigos e pesquisas sobre os problemas enfrentados por consumidores nos serviços de atendimento ao cliente das empresas, e a seguir foram investigadas as dificuldades das empresas em fornecer esses serviços de atendimento em vídeo, com qualidade e fácil acesso.

A partir do levantamento de requisitos a implementação do sistema foi iniciada através da plataforma *Node.js* e com a utilização dos frameworks *Express.js* e *Socket.io*, o primeiro com responsável pela API e o segundo com a tarefa de realizar conexões através de *WebSockets* (RINALDI, 2018).

O autor conclui que como resultado da natureza da aplicação, que envolvia diversas partes conectadas como: cliente, empresa, e atendente o projeto pode ser considerado multidisciplinar e com isso o trabalho proporcionou um aprendizado

importante sobre como arquitetar um sistema desde a visão geral até os detalhes da implementação.

6.5 COMUNICAÇÃO EM TEMPO REAL COM WEBSOCKET APLICADO AO DESENVOLVIMENTO DE UM PROTÓTIPO PARA PRESCRIÇÃO DE TREINO DE MUSCULAÇÃO PARA DISPOSITIVOS MÓVEIS ANDROID

Este é um trabalho de conclusão de curso que tem como autora Juliana Chaves Braz da Silva, apresentado para obtenção do grau de Bacharel no curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, no ano de 2015. O trabalho tem como objetivo a implementação de um protótipo de aplicativo para dispositivos móveis, utilizando WebSocket para a integração e comunicação em tempo real com aplicações web e *mobile* visando automatizar a prescrição de treinos de musculação em academias (SILVA, 2015).

Na área computacional do trabalho a autora descreve a plataforma Andriod, com seu recursos e ferramentas, aa plataforma web e suas características, os meios de conexão entre aplicações web e Andriod, além das tecnologias e protocolos para a sincronização em tempo real.

A autora conclui ao fim do trabalho, que durante o período de testes do aplicativo, alguns usuários utilizaram a ferramenta, e apesar de dispositivos e fabricantes distintos, o protótipo manteve a boa funcionalidade e conexão esperadas, tendo como ressalvas dos instrutores da academia apenas melhorias em relação a funcionalidade e disponibilidade para a plataforma iOS.

7 PROTOTIPO DE APLICAÇÃO PARA COMUNICAÇÃO EM TEMPO REAL P2P ENTRE NAVEGADORES UTILIZANDO WEBRTC

Este trabalho consiste no desenvolvimento de um protótipo de aplicação para comunicação em tempo real peer-to-peer entre navegadores utilizando o WebRTC, com o auxílio de outras tecnologias como HTML5, JavaScript e Node.js.

Pretende-se obter um protótipo funcional capaz de estabelecer comunicação direta entre navegadores por meio de uma página web utilizando o conceito P2P, além disso o protótipo deve possuir funcionalidades de transmissão de áudio, vídeo e texto empregando as APIs disponibilizadas pelo projeto WebRTC em conjunto com a sinalização de mensagens apoiada por um servidor Node.js.

7.1 METODOLOGIA

Para o desenvolvimento deste projeto de pesquisa foram instituídas as seguintes etapas metodológicas: levantamento bibliográfico, definição de ferramentas e recursos, analise de requisitos, prototipação de telas e desenvolvimento da aplicação.

Durante o levantamento bibliográfico foram estudados pontos específicos que auxiliaram na obtenção do conhecimento teórico necessário para o desenvolvimento do trabalho. Foram alvo de pesquisa os conceitos de P2P e RTC, além de um estudo mais detalhado do projeto WebRTC.

Na etapa seguinte consistiu no levantamento das ferramentas e recursos necessários para a implementação do protótipo, visando sempre a escolha de ferramentas e recursos populares atualmente e disponíveis de forma gratuita.

Na sequência foi realizada a análise de requisitos, onde foram definidas todas as funcionalidades e características do protótipo a ser desenvolvido e logo a seguir foi executado o processo de prototipagem das telas a serem construídas, definindo-se os layouts de tela do protótipo.

E por fim foi efetuado o processo de desenvolvimento, sendo o mesmo dividido em duas partes, a primeira parte se refere ao desenvolvimento do servidor de sinalização baseado em Node.js. Já a segunda parte ficou definida pela implementação da aplicação web, onde atua o WebRTC e suas funcionalidades.

7.2 DEFINIÇÃO DAS FERRAMENTAS E RECURSOS

Com base nas informações coletadas a partir do levantamento bibliográfico, em conjunto com as tendências das áreas de desenvolvimento envolvidas na concepção deste protótipo, foram escolhidas ferramentas gratuitas e que possam atender as demandas necessárias para a implementação do mesmo.

Para a criação dos diagramas de caso de uso e arquitetura do sistema foi escolhida a utilização da ferramenta online *LucidChart*, a mesma possui uma grande variedade de ferramentas para a criação de diversos tipos de diagramas incluindo os necessários para o planejamento do protótipo, além de possuir uma versão gratuita.

Para a concepção dos protótipos de tela do projeto, foi optado pela utilização da ferramenta *Balsamiq Mockups 3*. Esta ferramenta é bastante conhecida e disponibiliza uma grande quantidade de recursos, atendendo bem as necessidades do processo de prototipação de telas do projeto. A mesma possui uma versão gratuita por 30 dias, porém é possível conseguir a ferramenta gratuitamente como estudante, entrando em contato por e-mail com a empresa *Balsamiq* e fazendo uma solicitação.

Como linguagem de programação, foi definido o uso do JavaScript, devido principalmente ao fato de que os componentes WebRTC são acessados por meio de APIs JavaScript e também ao uso da plataforma Node.js na criação do servidor de sinalização. Isso torna possível o uso da mesma linguagem de programação tanto do lado do cliente como do servidor facilitado em parte o desenvolvimento do protótipo.

Para o desenvolvimento do servidor de sinalização, foi optado pelo uso da plataforma Node.js, que devido as suas características, as aplicações baseadas sobre a mesma possuem vantagens como o uso da linguagem JavaScript, alta escalabilidade e disponibilidade a um grande conjunto de APIs, com destaque a API do Socket.io.

A API do Socket.io em especial é de extrema importância para a funcionalidade do servidor de sinalização, uma vez que a mesma é responsável principalmente por desempenhar a função de transmissão e direcionamento de mensagens de sinalização, além do gerenciamento das salas de conversação e dos seus respectivos usuários.

No front-end foi definida a utilização do HTML5, uma vez que o mesmo disponibiliza recursos como as *tags* de áudio e vídeo que em conjunto com uma série

de atributos particulares a cada uma delas, permitem a execução das mídias capturadas pelo WebRTC.

Para auxiliar o HTML5 na construção do front-end, foi optado pela utilização do *framework Bootstrap 4*, sendo que o mesmo possui um grande número de classes e estilos em CSS em conjunto com algumas funcionalidades em JavaScript, disponibilizando ao usuário uma interface mais amigável e capaz de se adaptar a diversos tipos de dispositivos.

E por fim para a criação e edição dos códigos tanto em JavaScript como HTML5 e CSS utilizados na implementação do protótipo, foi definido a utilização do editor de texto *Sublime Text 3*.

A escolha do mesmo foi baseada no fato de o editor possuir uma extensa gama de plug-ins que podem ser adicionados ao mesmo, facilitando a escrita dos códigos em JavaScript e HTML5, e a utilização de classes do *Bootstrap 4*, além de possuir funcionalidades que facilitam e agilizam a criação e edição dos códigos desenvolvidos.

7.3 LEVANTAMENTO DE REQUISITOS

Segundo Van Lamsweerde (2009, tradução nossa), a engenharia de requisitos desempenha um papel de grande importância durante o desenvolvimento de software, tendo em vista que o sucesso ou falha do mesmo depende quase que integralmente da fase de engenharia de requisitos

Ainda de acordo com o autor, a engenharia de requisitos comumente é constituída por dois tipos distintos de requisitos: os requisitos funcionais (RF) que definem os recursos do software a ser desenvolvido e os requisitos não funcionais (RNF) que definem os atributos de qualidade desses recursos.

Para o levantamento dos RF, foram levadas em consideração as funcionalidades comuns as aplicações de comunicação em tempo real, como por exemplo realizar chamada de vídeo, chamada de voz e envio de mensagem de texto

Já a definição dos RNF, foi baseada em um estudo sobre aplicações web modernas, e as tecnologias utilizadas no desenvolvimento das mesmas, com a exceção da escolha do WebRTC, já que este foi um dos objetos de estudo da pesquisa desde o princípio.

Com base na pesquisa os RNF definidos são os seguintes:

a) RNF 01: Funcionar a partir de um navegador web;

b) RNF 02: Realizar comunicação P2P entre navegadores;

c) RNF 03: Utilizar linguagem JavaScript;

d) RNF 04: Limitar o número de participantes por sala a no máximo cinco;

e) RNF 05: Tornar obrigatório os campos de nome do usuário e sala;

f) RNF 06: Permitir a comunicação entre múltiplos pares simultaneamente;

g) RNF 07: Alertar a entrada e saída dos participantes da sala;

h) RNF 08: Alertar não possibilidade de transmissão de áudio e vídeo pelo navegador;

Com base aplicações de comunicação em tempo real existentes no mercado, em conjunto com as funcionalidades propostas para o protótipo foram definidos os seguintes RF:

a) RF 01: Fazer login;

b) RF 02: Criar sala;

c) RF 03: Acessar sala;

d) RF 04: Trocar de sala;

e) RF 05: Sair da sala;

f) RF 06: Habilitar Stream de vídeo;

g) RF 07: Desabilitar Stream de vídeo;

h) RF 08: Habilitar Stream de áudio;

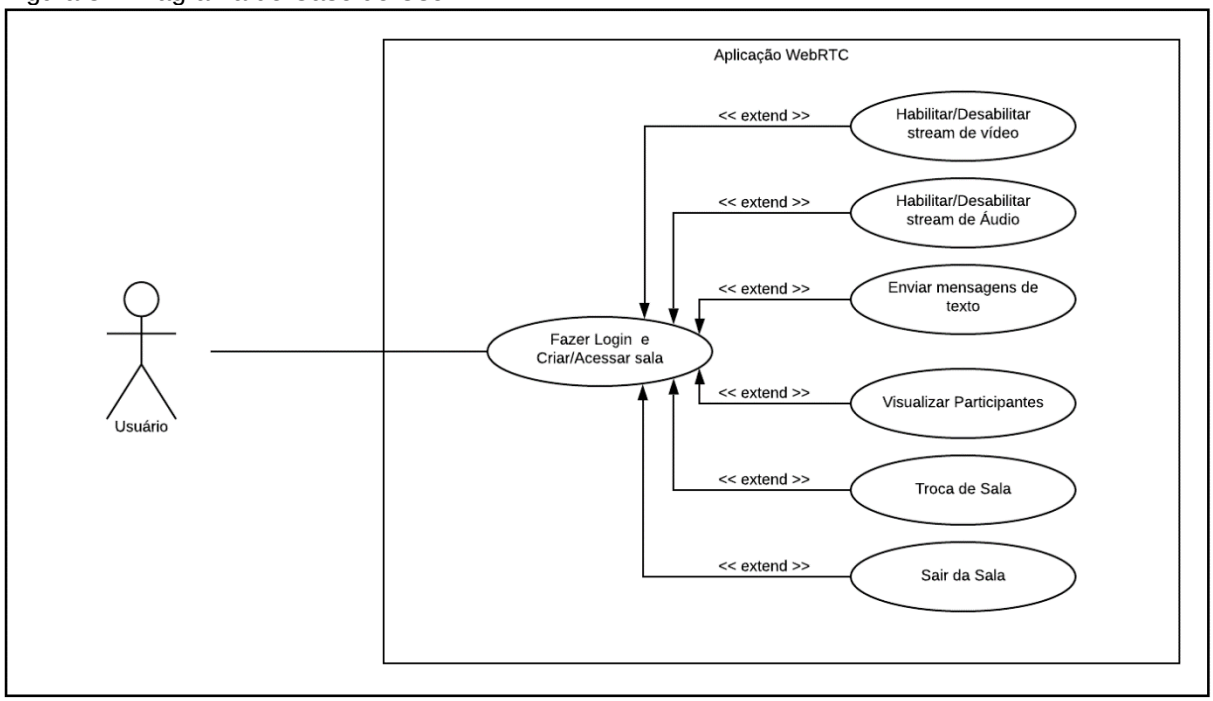
i) RF 09: Desabilitar Stream de áudio;

j) RF 10: Enviar mensagens de texto;

k) RF 11: Visualizar Participantes da sala;

Também com base nas funcionalidades propostas para o protótipo, foi constituído o diagrama de caso de uso mostrado na figura 8.

Figura 8 – Diagrama de Caso de Uso



Fonte: Do autor.

Como pode ser visto na figura anterior todas as funcionalidades do protótipo estendem a partir do momento em que o usuário faz o login e cria ou acessa as salas, tornando assim o login fundamental da aplicação.

7.4 PROTÓTIPOS DE TELA

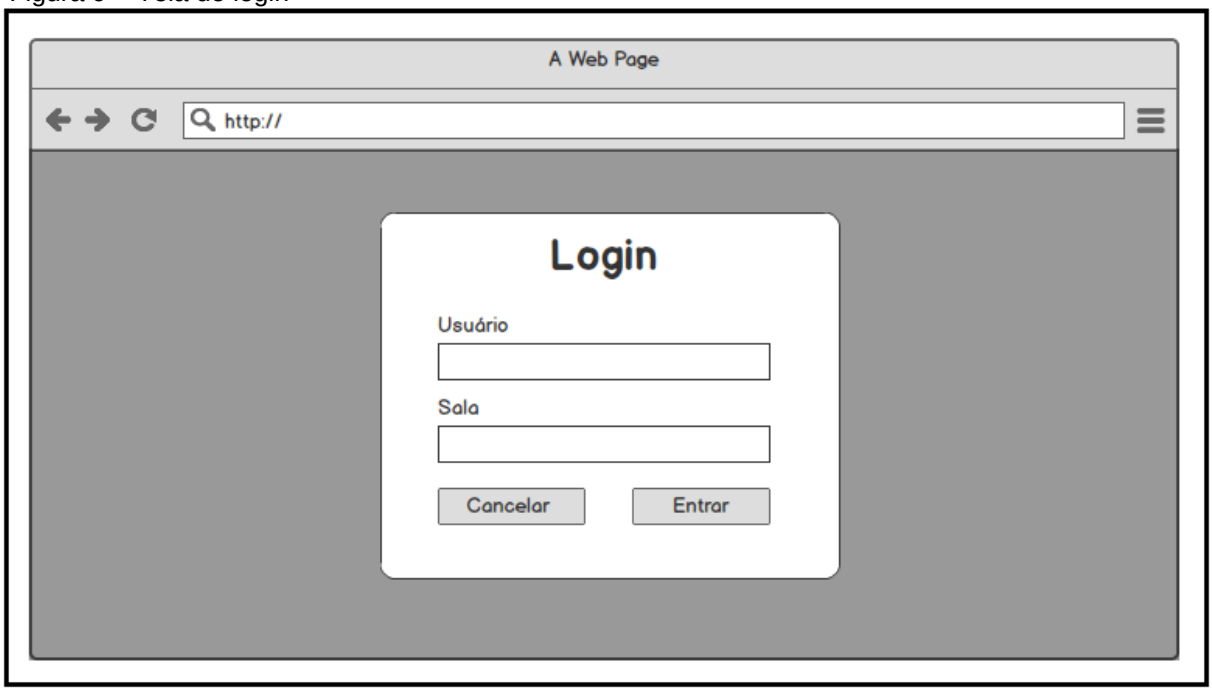
7.4.1 Tela de login

A tela de login possui dois campos do tipo texto, o primeiro para informar o nome do usuário e o segundo para informar o nome da sala que se deseja acessar ou criar.

Posteriormente existem dois botões distintos, o botão de cancelar, o qual o próprio nome sugere, cancela a ação do login fechando a modal responsável pelo o mesmo. Já o botão entrar é responsável por coletar as informações e enviá-las ao servidor, este fica responsável por tomar as decisões necessárias.

Caso a sala em questão não exista o servidor tem como comportamento criar uma sala com o nome definido pelo usuário, caso contrário o servidor adiciona o usuário a sala já existente e que possui o nome correspondente ao informado, tornando o usuário um novo participante da mesma, como mostrado na figura 9.

Figura 9 – Tela de login



Fonte: Do autor.

A tela de login é caracterizada por um layout bem simples, conforme pode ser verificado na figura anterior.

7.4.2 Tela da sala de conversação

A tela de conversação é a principal tela da aplicação e é composta por vários componentes e funcionalidades, a mesma é acessada logo após a tela de login e através dela é possível interagir com os demais integrantes da sala, como mostrado na figura 10.

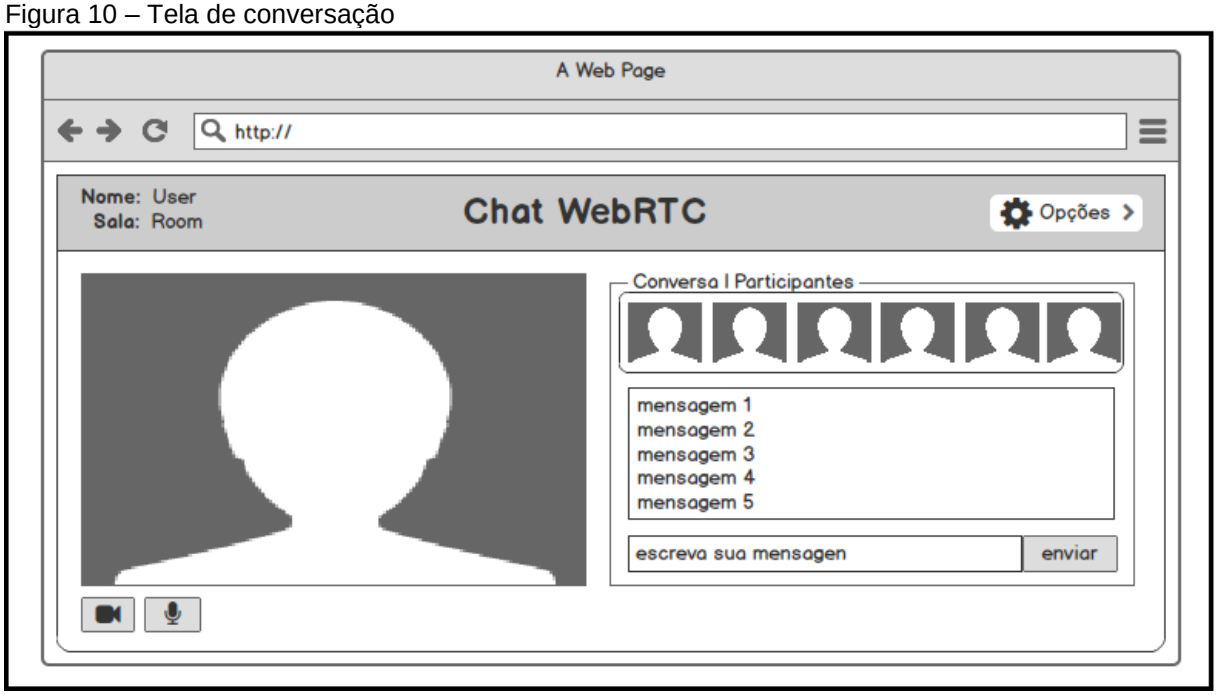
Esta tela da aplicação mostra na sua parte superior esquerda o nome do usuário e da sala a qual o mesmo está participando no momento. Na parte superior central fica o nome do protótipo e na parte superior esquerda localiza-se o botão de opções, o mesmo possui funcionalidades importantes para o protótipo, nele é possível

habilitar os canais de áudio e/ou vídeo assim como trocar ou sair da sala de que está participando.

Na parte esquerda central é mostrado o vídeo principal da conversa, o mesmo mostra o vídeo capturado pela câmera do usuário da aplicação caso o canal de vídeo esteja habilitado e o periférico de vídeo esteja ligado. Caso somente o canal de áudio esteja habilitado o quadro responsável pelo vídeo ficara visível, porém, sem nenhuma interatividade.

Na parte esquerda inferior estão botões que tem como finalidade gerenciar a atividade dos periféricos em relação a aplicação, os mesmos têm a função de ligar e desligar a webcam ou microfone do usuário, controlando o acesso que os demais participantes têm sobre a mídia gerada por cada um deles.

Figura 10 – Tela de conversação



Fonte: Do autor.

Na parte direita a tela é composta por um painel com duas abas, as quais são intituladas conversa e participantes, na composição da aba conversa na parte superior ficam disponíveis os streams de vídeo dos demais participantes da sala.

Logo abaixo fica localizado o histórico de mensagens trocadas pelo grupo, onde é possível visualizar todas as mensagens enviadas, desde a entrada do usuário na sala. E por fim existe um campo de texto e um botão para que o usuário possa enviar suas mensagens e interagir com os demais participantes.

Já a aba participantes é composta por uma tabela simples, sendo que a mesma possui duas colunas de informações. A primeira coluna mostra os *IDs* dos usuários e em seguida a segunda coluna disponibiliza o nome dos mesmos.

Esta lista é atualizada sempre que um participante interage saindo ou entrando da sala. O controle dos *IDs* e nomes dos usuários é responsabilidade do servidor que gerencia os mesmos a cada login na aplicação.

7.5 ARQUITETURA DO PROTÓTIPO

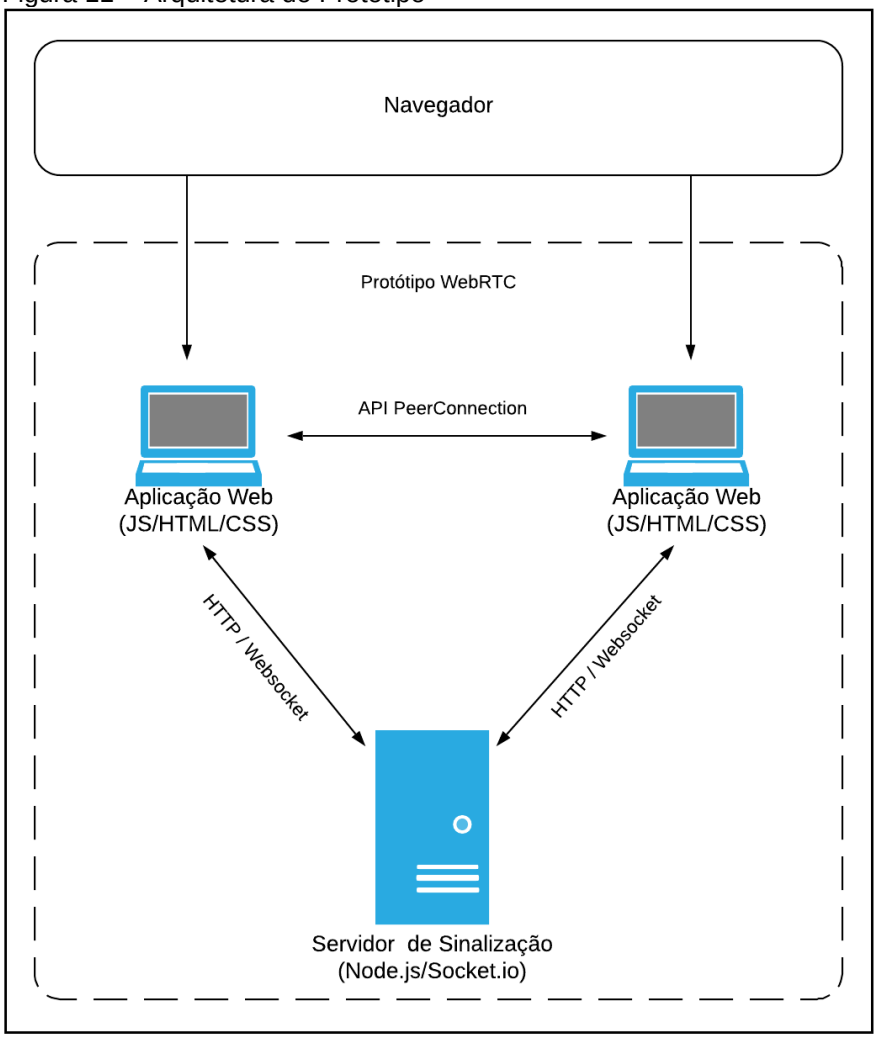
A arquitetura do protótipo desenvolvido respeita o modelo de triângulo proposto pelo WebRTC e descrito anteriormente durante a fundamentação teórica. Este modelo de arquitetura diferentemente do modelo de trapézio, é composto por um único servidor de sinalização, e o mesmo se conecta aos seus clientes por meio de conexões do tipo HTTP/WebSocket.

O servidor de sinalização, como o nome sugere tem o papel de sinalizar a presença de clientes que estejam dispostos a estabelecer uma conexão P2P com os demais participantes. Além disso o mesmo fica responsável pelas negociações de oferta, resposta e candidatos, transmitindo durante este processo, os parâmetros necessários para a criação de uma conexão direta entre os clientes envolvidos.

As aplicações Web do lado do cliente, ao se conectarem com o servidor de sinalização, em primeiro lugar comunicam aos demais clientes sobre sua presença, e em seguida, caso haja outros pares disponíveis para uma possível comunicação P2P as mesmas disponibilizam ao servidor de sinalização suas credenciais e seus protocolos de comunicação.

No caso de sucesso nas negociações entre as aplicações dos clientes, as mesmas criam uma conexão direta entre elas, utilizando a API PeerConnection presente no WebRTC, e a partir deste momento são capazes de gerenciar canais de comunicação possibilitando a transmissão de faixas de áudio, vídeo e texto. Esta arquitetura está representada na figura 11.

Figura 11 – Arquitetura do Protótipo



Fonte: Do autor.

Na figura anterior é possível se ter uma visão geral da arquitetura proposta e como funciona o fluxo de informações por entre os componentes descritos anteriormente.

7.6 DESENVOLVIMENTO DO SERVIDOR DE SINALIZAÇÃO

Durante o estudo e desenvolvimento do protótipo proposto, foi possível identificar que alguns trechos de implementação são de suma importância para as

funcionalidades do protótipo quando se trata do servidor de sinalização. A seguir serão apresentadas as funções ou o conjunto delas cuja a importância é vital desenvolvimento da aplicação.

7.6.1 Definição de Bibliotecas

Para a criação do servidor de sinalização na plataforma Node.js foi necessária a utilização de algumas bibliotecas, visando agilizar o processo de desenvolvimento e tornando o código mais claro, porém, para que estas bibliotecas possam ser utilizadas, precisam ser referenciadas no código fonte como visto na figura 12.

Figura 12 – Definindo o uso das bibliotecas Node.js

```
var nodeStatic = require('node-static');
var https      = require('https');
var fs         = require('fs');
var socketIO   = require('socket.io');
```

Fonte: Do autor.

A biblioteca *node-static* tem como funcionalidade permitir que o servidor de sinalização possa trabalhar também como servidor estático, servindo assim os arquivos necessários para a aplicação web necessária para o chat.

A biblioteca *https* é utilizada em conjunto com a biblioteca *fs*, a primeira traz as funções necessárias para a criação de um servidor com o protocolo *HTTPS,* já a segunda é utilizada para referenciar os arquivos dos certificados necessários para que o servidor *https* funcione.

Por fim a biblioteca *socket.io* traz consigo um conjunto de funções bastante completo para a implementação de um servidor de sinalização, disponibilizando funções para o gerenciamento de salas de conversação e seus usuários, além de um serviço de envio de mensagens inspirado em *WebSockets,* utilizados para o envio das mensagens de sinalização.

7.6.2 Criação do Servidor de Sinalização

A criação de um servidor na plataforma Node.js é de fácil implementação e a partir de algumas linhas de código é possível criar um servidor funcional, como mostrado na figura 13.

Figura 13 – Criação do servidor

```
var fileServer = new(nodeStatic.Server)();
var app = https.createServer(options, function(req, res) {
    res.setHeader('Feature-Policy', "camera *; microphone *");
    fileServer.serve(req, res);
}).listen(21130, function(){
    console.log('Ouvindo na porta: 21130');
});

var io = socketIO.listen(app);
```

Fonte: Do autor.

Na figura 13 é possível ver como é feita a implementação da criação do servidor, partindo da utilização da biblioteca *node-static* para a importação de métodos para o servidor estático.

A seguir utiliza-se a função *createServer* da biblioteca *https* para a criação do servidor e durante a execução desta, é definido o cabeçalho *HTTPS* para a política de recursos, permitindo o uso de câmeras e microfones a partir das páginas disponibilizadas pelo servidor. Já na sequência o servidor é definido como estático se utilizando dos métodos importados anteriormente.

Após a criação do servidor, o mesmo passa a escutar na porta definida como parâmetro da função *listem,* que quando executada acaba por mostrar no console do servidor uma mensagem. Logo após é utilizada a função *listem* da biblioteca *socket.io,* esta utiliza o servidor como parâmetro e tem como função escutar todos os eventos disparados pela biblioteca *socket.io* utilizada do lado do cliente, ou seja, pela aplicação web.

7.6.3 Gerenciamento das Salas de Bate-Papo

O gerenciamento das salas de bate-papo é parte fundamental para o funcionamento do servidor de sinalização, uma vez que estas funções são

responsáveis por criar salas e delegar as ações a serem tomadas em relação aos seus possíveis candidatos.

A partir do momento em que os usuários se encontram devidamente separados por salas de conversação, torna-se viável o envio de mensagens de sinalização, pois cada integrante da sala é capaz de reconhecer os emissores e receptores disponíveis para eventuais trocas de mensagens de sinalização.

A figura 14 mostra a implementação desenvolvida para a entrada dos participantes nas salas de bate papo.

Figura 14 – Entrada na sala de Bate-Papo

```
socket.on('entrarSala', function(room, name) {
    var listaPeers = io.sockets.adapter.rooms[room];
    var numPeers = listaPeers ? Object.keys(listaPeers.sockets).length : 0;

    //*********************************************************//

    listaUsers.push({id:socket.id, name:name});

    //*********************************************************//

    if (numPeers === 0) {
        socket.join(room);
        socket.emit('criouSala', room, socket.id);
    } else if (numPeers <= 5) {
        socket.join(room);
        io.in(room).emit('listaPartic', listaUsers);

        //*********************************************************//

        for (var socketId in listaPeers.sockets) {
            if (socketId != socket.id) {
                socket.emit('criandoConexao', room, socketId);
            }
        }

        //*********************************************************//

        socket.emit('entrouSala', room, socket.id);

        //*********************************************************//

        socket.broadcast.to(room).emit('peerPronto', room, socket.id);

    } else {
        socket.emit('salaCheia', room);
    }
});
```

Fonte: Do autor.

No código mostrado pela figura 14, inicialmente é possível visualizar a utilização da biblioteca *socket.io* para a implementação da função *entrarSala.* O socket.io no servidor espera o disparo do evento do lado do cliente, trazendo como parâmetros para a função o nome da sala e o nome do usuário que deseja criar ou participar da mesma.

Durante a execução desta função encontra-se uma estrutura de decisão que possui três condições e cada uma delas é responsável por executar um conjunto de ações diferentes.

Se o número de pares conectados a sala for igual a zero, é considerado fato que a sala em questão não existe, então é chamada a função *join* que é responsável por criar a sala e adicionar o usuário na mesma.

A condição a seguir, serve primeiramente para limitar a no máximo cinco, o número de participantes da sala, por isso a mesma consiste em verificar se o número de pares é menor ou igual a cinco.

Caso a condição seja verdadeira é chamada a função *join* para que o usuário seja integrado a sala informada, logo após é emitida para todos os usuários da sala, a lista de participantes atualizada.

A seguir é emitida uma mensagem para o cliente que acaba de entrar na sala, solicitando que o mesmo crie uma conexão com os demais participantes, passando como parâmetros o nome da sala e o identificador de cada participante.

Terminado o envio de mensagens para a criação de conexões é emitida mais uma mensagem informando o nome da sala e o identificador do par que acaba de entrar na sala. E por fim é enviada uma mensagem para todos os pares da sala exceto o que acaba de entrar, confirmando que o mesmo está disponível para possíveis conexões P2P.

Se nenhuma das condições anteriores for atendida, então, significa que a sala já atingiu o número máximo de participantes, então, é retornada uma mensagem para o par que desejava conectar-se, avisando que a sala está cheia.

7.6.4 Envio de Mensagens de Sinalização

O envio de mensagens de sinalização é executado por uma função bastante sucinta, que é chamada a partir do evento *enviarSinal* disparado pela

aplicação web em direção ao servidor. Esta função recebe como parâmetros a mensagem a ser enviada e o identificador do destinatário, como pode ser visto na implementação mostrada na figura 15.

Figura 15 – Envio de mensagens de sinalização.

```
socket.on('enviarSinal', function(message, destino) {
    io.to(destino).emit('receberSinal', message, socket.id);
});
```

Fonte: Do autor.

A função consiste basicamente em enviar uma mensagem para o destino solicitando que este receba o sinal, passando como parâmetros a mensagem e o identificador do par remetente.

7.7 DESENVOLVIMENTO DA APLICAÇÃO WEB

A aplicação web é a responsável pelas funcionalidades disponibilizadas aos clientes, isso desde o momento da conexão, e permanecendo durante todo o período em que o usuário estiver conectado.

Para a implementação desta aplicação, foi necessário a utilização de uma extensa gama de funções as quais tornam a aplicação mais interativa e funcional, porém, entre todas as funções serão destacas as que fazem uso das APIs do WebRTC uma vez que o mesmo é um dos objetos principais desta pesquisa.

7.7.1 Criação de Conexões P2P WebRTC e Envio de Mensagens Sinalização

Para que todos os integrantes sala de bate-papo pudessem interagir entre si, foi necessário o desenvolvimento de uma função chamada *criarPeerConnection* capaz de conectar todos os pares uns aos outros, criando uma estrutura semelhante a uma rede *mesh*.

Todas as conexões criadas são colocadas em uma variável do tipo *Array* criando uma espécie de lista de conexões, e para cada uma delas é atribuído um número identificador, sendo este, o mesmo número identificador do par remoto o qual se deseja estabelecer a conexão P2P.

A partir do momento em que são criadas as conexões, começam os trâmites de negociação entre os pares, mas, para que fiquem definidas, cada uma das ações a serem tomadas durante as negociações, a função *criarPeerConnection* recebe como um dos parâmetros a variável *isCriador,* definindo se o par que deseja se conectar deve ofertar uma possível conexão e criar um canal de dados para a comunicação ou apenas participar de um canal de dados previamente criado por outro par. A figura 16 mostra a parte da função responsável por esta tomada de decisão.

Figura 16 – Criação de Oferta de Negociação

```
if (isCriador) {
    dataChannel[idConn] = peerConn[idConn].createDataChannel('message');
    onDataChannelCreated(dataChannel[idConn], idConn);

    peerConn[idConn].createOffer().then(function(offer) {
        return peerConn[idConn].setLocalDescription(offer);
    }).then(function() {
        enviarSinal(peerConn[idConn].localDescription, idConn);
    });
} else {
    peerConn[idConn].ondatachannel = function(event) {
        dataChannel[idConn] = event.channel;
        onDataChannelCreated(dataChannel[idConn], idConn);
    };
}
```

Fonte: Do autor.

A função *criarPeerConnection* ainda pode ser subdividida pela utilização de alguns *EventListeners* nativos do objeto *RTCPeerConnection,* e que são de suma importância para que se consiga estabelecer uma conexão multimídia e direta entre os pares.

O *EventListener* chamado *onicecandidate,* é responsável por identificar o evento de ingresso de um candidato a conexão, e a partir deste evento foi implementado uma função para o envio de uma mensagem de sinalização, contendo os parâmetros do candidato em questão, para que possa ser estabelecida uma possível conexão, como pode ser visto na figura 17.

Figura 17 – Utilização de *onicecandidate*

```
peerConn[idConn].onicecandidate = function(event) {
    if (event.candidate && peerConn[idConn].signalingState == 'stable') {
        enviarSinal({
            type: 'candidate',
            sdpMLineIndex: event.candidate.sdpMLineIndex,
            sdpMid: event.candidate.sdpMid,
            candidate: event.candidate.candidate
        }, idConn);
    } else {
        console.log('Fim dos Candidatos');
    }
};
```

Fonte: Do autor.

Ainda existem mais dois *EventListeners* nativos do objeto *RTCPeerConnection* que são utilizados dentro da função *criarPeerConnection* e são estes chamados de *onnegotiationneeded e ontrack* e suas respectivas utilizações podem ser vistas a partir da figura 18.

Figura 18 – Utilização de *onnegotiationneeded e ontrack*

```
peerConn[idConn].onnegotiationneeded = function() {
    if (peerConn[idConn].videoTrack || peerConn[idConn].audioTrack) {
        peerConn[idConn].createOffer().then(function(offer) {
            return peerConn[idConn].setLocalDescription(offer);
        }).then(function() {
            console.log('Envia descrição local');
            enviarSinal(peerConn[idConn].localDescription, idConn);
        });
    }
};

peerConn[idConn].ontrack = function (e) {
    console.log('Adiciona a webCam e Microfone a conexão');

    if (e.track.kind === 'video') {
        $('#divVideos').append('<video id="video_'+idConn+'" width="130" '+
            'class="border rounded" autoplay playsinline></video>');
        document.getElementById('video_'+idConn).srcObject = e.streams[0];
    } else {
        $('#divAudios').append('<audio id="audio_'+idConn+'" width="130" '+
            'class="border rounded" autoplay playsinline></audio>');
        document.getElementById('audio_'+idConn).srcObject = e.streams[0];
    }
};
```

Fonte: Do autor.

O *EventListener* chamado *onnegotiationneeded* tem a função de verificar a necessidade de uma nova negociação, então, no caso de uma conexão de áudio ou vídeo ser solicitada, a função desenvolvida decide criar uma nova oferta de conexão,

repetindo todos os procedimentos de oferta e resposta, assim atualizando os parâmetros da conexão já existentes.

Já o *ontrack* captura o evento de adição de faixa de mídia, no caso de uma conexão de sucesso utilizando áudio ou vídeo, então, na implementação proposta é verificado o tipo de faixa, e, a partir deste momento são adicionadas as mídias a tela da aplicação, podendo estas serem faixas de áudio ou *streams* de vídeo.

7.7.2 Recebimento de Mensagens de Sinalização e Envio de Resposta

Pode-se definir como parte fundamental para o estabelecimento de uma conexão P2P utilizando WebRTC, não somente a criação e o envio de mensagens de sinalização, mas, também o recebimento e envio de resposta, uma vez que o receptor e a resposta são necessários para que haja comunicação de fato. A figura 19 mostra o trecho de código responsável por receber as mensagens de sinalização.

Figura 19 – Recebimento de Mensagens de Sinalização

```
function getRetornoSinal(msg, clientId) {
    if (msg.type === 'offer') {
        peerConn[clientId].setRemoteDescription(new RTCSessionDescription(msg));
        peerConn[clientId].createAnswer().then(function(answer) {
            return peerConn[clientId].setLocalDescription(answer);
        }).then(function() {
            enviarSinal(peerConn[clientId].localDescription, clientId);
        })

    } else if (msg.type === 'answer') {
        peerConn[clientId].setRemoteDescription(new RTCSessionDescription(msg));

    } else if (msg.type === 'candidate') {
        peerConn[clientId].addIceCandidate(new RTCIceCandidate(msg));
    }
}
```

Fonte: Do autor.

O recebimento de mensagens de sinalização é efetuado pela função *getRetornoSinal* sendo que a mesma recebe dois parâmetros, o primeiro é a mensagem e o segundo o identificador do remetente, e, a partir do tipo de mensagem serão definidas as ações a serem tomadas.

Caso a mensagem seja do tipo *offer*, ou seja, uma mensagem de oferta, isso representa que o remetente deseja estabelecer uma comunicação com o

destinatário, para isso, são adicionados os parâmetros do remente a conexão estabelecida, e a seguir é criada a resposta, que, por sua vez, passa os parâmetros do destinatário de volta ao remetente por meio de uma mensagem de sinalização.

Se o tipo da mensagem for *answer,* ou seja, mensagem de resposta, esta é obtida a partir da resposta em relação a uma oferta de comunicação, então, a ação tomada, resume-se em adicionar as configurações do remetente na conexão proposta. Já caso a mensagem seja do tipo *candidate* significa que foi informado um candidato a conexão, e então as credenciais do candidato são adicionadas a conexão.

7.7.3 Acessando Entradas Multimídias com WebRTC

Uma das características mais fascinantes do WebRTC, além de conseguir realizar conexões P2P entre navegadores, é o fato de poder gerenciar entradas multimídia e retransmitir os dados coletados, diretamente pelo navegador.

Com esse intuito, durante a implementação do protótipo foram desenvolvidas duas funções bastante semelhantes uma com objetivo de captar e gerenciar vídeo e outra áudio. Uma dessas funções pode ser vista na figura 20.

Figura 20 – Iniciando *stream* de vídeo

```
async function iniciaVideo() {
    try {
        streamVideo = await navigator.mediaDevices.getUserMedia({video: true});

        streamVideo.getTracks().forEach(function (track) {
            $.each(connections, function(index, idConn) {
                track.enabled = false;
                peerConn[idConn].addTrack(track, streamVideo);
                peerConn[idConn].videoTrack = true;
            });
        });

        document.getElementById('video1').srcObject = streamVideo;
        $('#btnVideo').prop('disabled', false);
        $('#divStream').removeClass('d-none');

    } catch (e) {
        console.log(e);
        mostrarAlerta('Seu Navegador Não Pode transmitir Vídeo');
    }
}
```

Fonte: Do autor.

A função *iniciaVideo* tem como objetivo verificar a possibilidade da adição de um canal de vídeo a uma conexão previamente estabelecida, para isso, a mesma começa utilizando o comando *navigator.mediaDevices.getUserMedia* que solicita ao usuário o acesso as mídias de entrada como microfone e câmera.

A seguir o método *getTracks* busca todas as mídias capturadas e em seguida utiliza-se o método *addTrack*, do objeto *RTCPeerConnection*, fazendo com que todas as conexões existentes adicionem o canal de mídia informado.

Caso não haja nenhuma entrada multimídia disponível, isso acaba forçando a função *iniciaVideo* a emitir uma mensagem ao usuário informando que o mesmo não pode transmitir a mídia solicitada.

7.8 RESULTADOS OBTIDOS

A partir das pesquisas sobre a arquitetura P2P, conceito de RTC e dos levantamentos bibliográficos e estudos sobre Node.js, Socket.io e APIs do projeto WebRTC, foi possível a obtenção de uma coletânea de estudos bastante rica.

Este processo de estudo, pesquisa e coleta de materiais, foi parte importante para desenvolvimento do protótipo proposto, uma vez que para a implementação do mesmo foram necessários conhecimentos mais aprofundados sobre as tecnologias empregadas.

Durante as pesquisas foram encontrados alguns trabalhos que tinham o WebRTC como objeto de estudo, porém em nenhuma das implementações foi encontrada a utilização do WebRTC na forma de multiconexões, sendo que o mesmo foi projetado para a comunicação par a par e não multipares.

A implementação do WebRTC para comunicação multipares, causou algumas dificuldades durante o processo de desenvolvimento, em um primeiro momento pela inexperiência do desenvolvedor em relação a tecnologia, e posteriormente pelo fato de a lógica utilizada para a criação de multiconexões não possuir nenhuma referência previa.

Durante os testes com o protótipo pode se encontrar um ponto fraco em relação ao modo de multiconexões inspirado em redes *mesh*, especialmente durante as transmissões multimídia, principalmente pelo fato de que, o número de conexões crescia proporcionalmente em relação ao número de usuários, exigindo muito da

largura de banda da conexão. Porém como solução para este problema, foi decidido por limitar o número de integrantes à cinco por sala, resultando no máximo de quatro conexões por usuário.

Após o processo de desenvolvimento e testes, obteve-se um protótipo funcional, com o poder de gerenciar o uso de entradas multimídia, como microfones e webcans diretamente pelo navegador. Além disso o protótipo é capaz transmitir em tempo real, texto, áudio e vídeo em tempo real, diretamente entre dois ou mais navegadores, não importando onde os mesmos estejam.

Todo o processo de desenvolvimento foi devidamente documentado e o protótipo em questão pode servir como base para o desenvolvimento de futuras aplicações, uma vez que esse tipo de aplicação pode ser útil, não só para fins recreativos, mas também para empresas que necessitam de soluções personalizadas a um custo considerado baixo.

Após a conclusão de todas as etapas propostas, os resultados obtidos foram satisfatórios, uma vez que todos os objetivos específicos descritos anteriormente foram alcançados. Este trabalho de conclusão de curso tinha como objetivo geral o desenvolvimento de um protótipo para comunicação em tempo real Peer-to-Peer entre navegadores utilizando WebRTC, e conforme os testes realizados o mesmo pode ser concluído.

8 CONCLUSÃO

Durante o desenvolvimento deste trabalho, ficou evidente a importância de serem desenvolvidas novas tecnologias para a comunicação em tempo real, uma vez que as aplicações desse gênero, atualmente, estão baseadas na arquitetura cliente/servidor e são disponibilizadas por grandes empresas, assim as mesmas estão sujeitas a possíveis falhas, cobranças e até vazamento de informações.

Para o entendimento e desenvolvimento do protótipo proposto, foi necessário um levantamento bibliográfico bastante sólido, partindo desde os conceitos da arquitetura P2P e RTC, até o estudo de tecnologias para desenvolvimento web, e sobre o projeto WebRTC, construindo assim a base de conhecimento necessária para a concepção do protótipo.

A metodologia proposta para o desenvolvimento do protótipo, foi baseada em assuntos estudados anteriormente, como, engenharia de software, analise de requisitos e desenvolvimento web. A mesma em conjunto com o levantamento bibliográfico, foram responsáveis por permitir que tanto o objetivo geral quanto os objetivos específicos fossem alcançados, resultando assim, em um protótipo de aplicação para comunicação em tempo real P2P entre navegadores.

Devido à natureza do projeto WebRTC, durante o processo de implementação do protótipo, foi necessário o desenvolvimento de uma lógica de programação capaz de fazer comunicação entre vários pares, inspirada na tipologia de redes *mesh*, uma vez que o WebRTC que foi desenvolvido para fazer comunicação par a par e não multipares.

Durante os testes, a comunicação entre vários pares demonstrou um ponto fraco sendo este, o fato de que o número de conexões aumentar proporcionalmente

ao número de usuários, causando um consumo cada vez maior da largura de banda disponível, principalmente durante as comunicações por stream de vídeo. Sendo assim foi necessária a limitação para no máximo de cinco, o número de integrantes para cada sala de bate-papo.

Ao decorrer do desenvolvimento deste trabalho, foram encontrados alguns tópicos os quais são passiveis de possíveis melhoramentos, permitindo assim a extensão do projeto para novas pesquisas na área e trabalhos futuros. Os tópicos de maior destaque são:

a) Desenvolver uma biblioteca em JavaScript, inspirada na lógica desenvolvida para a conexão multipares, a fim de padroniza-la;
b) Pesquisar e desenvolver métodos para a diminuição do consumo da largura de banda, principalmente durante os *streams* de vídeo;
c) Estudar métodos de envio de arquivos para vários pares, uma vez que a biblioteca WebRTC permite a troca de arquivos par a par;

No entanto, podemos concluir que o trabalho desenvolvido pode atender todas as expectativas, alcançando os objetivos propostos no início do projeto, a partir da obtenção de um protótipo de aplicação para comunicação em tempo real entre navegadores utilizando WebRTC.

REFERÊNCIAS

ABUDOULIKEMU, Yimamuaishan; HUANG, Yuanming; YE, Changqing. A Scalable intelligent service model for video surveillance system based on RTCP. **2010 2nd International Conference On Signal Processing Systems**, [s.l.], p.346-349, jul. 2010. IEEE. http://dx.doi.org/10.1109/icsps.2010.5555783.

AMIRANTE, Alessandro *et al.* On the seamless interaction between webRTC browsers and SIP-based conferencing systems. **Ieee Communications Magazine**, [s.l.], v. 51, n. 4, p.42-47, abr. 2013. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/mcom.2013.6495759.

AMMAR, Doreid. An Experimental Platform for QoE Studies of WebRTC-based Multi-Party Video Communication. **International Journal Of New Computer Architectures And Their Applications**, [s.l.], v. 8, n. 2, p.89-94, 2018. The Society of Digital Information and Wireless Communications (SDIWC). http://dx.doi.org/10.17781/p002437.

ATUCHUKWU, Charles. Web-based Instant Messaging. 2009. 9 f. Tese (Doutorado)-Curso de Computer Science, University Of The Western Cape, A, 2009. Disponível em:http://www.cs.uwc.ac.za/~acharles/documentation1.pdf. Acesso em: 28 out. 2018

BARRY, Bazara I. A.; TOM, Fatma M.. INSTANT MESSAGING: STANDARDS, PROTOCOLS, APPLICATIONS, AND RESEARCH DIRECTIONS. In: KUTAIS, B.g.. Internet Policies and Issues, Volume 7. A: Nova Science Publishers, Inc, 2009. Cap. 7. p. 1-14. Disponível em: https://www.researchgate.net/publication/280307922_Instant_Messaging_Standards_Protocols_Applications_and_Research_Directions. Acesso em: 28 out. 2018

BOROWSKY, E.; LOGAN, A.; SIGNORILE, R.. Leveraging the Client-Server Model in P2P: Managing Concurrent File Updates in a P2P System. **Advanced Int'l Conference On Telecommunications And Int'l Conference On Internet And Web Applications And Services (aict-iciw'06)**, [s.l.], p.1-6, 2006. IEEE. http://dx.doi.org/10.1109/aict-iciw.2006.123.

BOUBENDIR, Amina; BERTIN, Emmanuel; SIMONI, Noemie. Network as-a-service: The WebRTC case. **2015 6th International Conference On The Network Of The Future (nof)**, [s.l.], p.1-3, set. 2015. IEEE. http://dx.doi.org/10.1109/nof.2015.7333308.

BUFORD, John F.; YU, Heather; LUA, Eng Keong. **P2P Networking and Applications.** Burlington: Elsevier, 2009. 438 p.

CHEN, Li-li; LIU, Zheng-long. Design of Rich Client Web Architecture Based on HTML5. **2012 Fourth International Conference On Computational And**

Information Sciences, [s.l.], p.1009-1015, ago. 2012. IEEE. http://dx.doi.org/10.1109/iccis.2012.125.

CHITRA, Lakshmi Prasanna; SATAPATHY, Ravikanth. Performance comparison and evaluation of Node.js and traditional web server (IIS). **2017 International Conference On Algorithms, Methodology, Models And Applications In Emerging Technologies (icammaet)**, [s.l.], p.1-4, fev. 2017. IEEE. http://dx.doi.org/10.1109/icammaet.2017.8186633.

DAI, Bintao *et al.* HTML5-based interactive media editing system. **2017 3rd Ieee International Conference On Computer And Communications (iccc)**, [s.l.], p.1185-1189, dez. 2017. IEEE. http://dx.doi.org/10.1109/compcomm.2017.8322730.

DAVIDS, Carol; ORMAZABAL, Gaston; STATE, Radu. Real-time communications. **Acm Sigcomm Computer Communication Review**, [s.l.], v. 44, n. 3, p.112-115, 28 jul. 2014. Association for Computing Machinery (ACM). http://dx.doi.org/10.1145/2656877.2656894.

DESMET, Lieven; JOHNS, Martin. Real-Time Communications Security on the Web. **Ieee Internet Computing**, [s.l.], v. 18, n. 6, p.8-10, nov. 2014. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/mic.2014.117.

DETSCH, André; GASPARY, Luciano Paschoal; BARCELLOS, Marinho Pilla. **Uma Abordagem para Incorporação Flexível de Aspectos de Segurança em Aplicações Peer-to-Peer.** 2006. 16 f. Monografia (Especialização) - Curso de Pós-graduação em Computação Aplicada (pipca), Universidade do Vale do Rio dos Sinos (unisinos), São Leopoldo, 2006. Disponível em: https://www.researchgate.net/publication/255621789_Uma_Abordagem_para_Incorporacao_Flexivel_de_Aspectos_de_Seguranca_em_Aplicacoes_Peer-to-Peer. Acesso em: 05 abr. 2019.

DISTERHOFT, Andreas; GRAFFI, Kalman. **Protected chords in the web: secure P2P framework for decentralized online social networks**. 2015 Ieee International Conference On Peer-to-Peer Computing (p2p), [s.l.], p.1-2, set. 2015. IEEE. http://dx.doi.org/10.1109/p2p.2015.7328520.

EDÄNGE, Simon. **An Implementation and Performance Evaluation of a Peer-to-Peer Chat System.** 2015. 40 f. Tese (Doutorado) - Curso de Computer Science, Faculty Of Computing Blekinge Institute Of Technology, Karlskrona, 2015. Disponível em:https://pdfs.semanticscholar.org/390c/633afd8b2ed1908fcb18a75ea13069f8dc3c.pdf. Acesso em: 26 out. 2018.

ESHKEVARI, Laleh *et al.* JSDeodorant: Class-Awareness for JavaScript Programs. **2017 Ieee/acm 39th International Conference On Software**

Engineering Companion (icse-c), [s.l.], p.71-74, maio 2017. IEEE. http://dx.doi.org/10.1109/icse-c.2017.6.

FERNANDEZ, Luis Lopez *et al.* Kurento: a media server technology for convergent WWW/mobile real-time multimedia communications supporting WebRTC. **2013 Ieee 14th International Symposium On**, [s.l.], p.1-6, jun. 2013. IEEE. http://dx.doi.org/10.1109/wowmom.2013.6583507.

FERNANDEZ, Luis Lopez *et al.* Authentication, Authorization, and Accounting in WebRTC PaaS Infrastructures: The Case of Kurento. **Ieee Internet Computing**, [s.l.], v. 18, n. 6, p.34-40, nov. 2014. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/mic.2014.102.

GHAREEB, Majd *et al.* Client/Server and Peer-to-Peer hybrid architecture for adaptive video streaming. **2015 International Conference On Communications, Signal Processing, And Their Applications (iccspa'15)**, [s.l.], p.1-6, fev. 2015. IEEE. http://dx.doi.org/10.1109/iccspa.2015.7081293.

GROSSKURTH, A.; GODFREY, M.w.. A reference architecture for Web browsers. **21st Ieee International Conference On Software Maintenance (icsm'05)**, [s.l.], p.1-4, 2005. IEEE. http://dx.doi.org/10.1109/icsm.2005.13.

HALLARAKER, O.; VIGNA, G.. Detecting Malicious JavaScript Code in Mozilla. **10th Ieee International Conference On Engineering Of Complex Computer Systems (iceccs'05)**, [s.l.], p.1-10, 2005. IEEE. http://dx.doi.org/10.1109/iceccs.2005.35.

JAOUHARI, Saad El. **A secure design of WoT services for smart cities.** 2018. 220 f. Tese (Doutorado) - Curso de Networking And Internet Architecture, Ecole Nationale Supérieure Mines-télécom Atlantique, Rennes, 2019. Disponível em: https://www.researchgate.net/publication/332368566_A_secure_design_of_WoT_services_for_smart_cities. Acesso em: 18 jun. 2019.

JIANBING, Liang; SHUHUI, Chen. The Design and Implementation of RTSP/RTP Multimedia Traffic Identification Algorithm. **Journal Of Physics**: Conference Series, [s.l.], v. 1168, p.1-8, fev. 2019. IOP Publishing. http://dx.doi.org/10.1088/1742-6596/1168/5/052033.

JIBAJA, Ivan *et al.* Vector Parallelism in JavaScript: Language and Compiler Support for SIMD. **2015 International Conference On Parallel Architecture And Compilation (pact)**, [s.l.], p.407-418, out. 2015. IEEE. http://dx.doi.org/10.1109/pact.2015.33.

JIMENEZ, Jaime Galan; CERVERO, Alfonso Gazo. Overview and Challenges of Overlay Networks: A Survey. **International Journal Of Computer Science &**

Engineering Survey, [s.l.], v. 2, n. 1, p.19-37, 28 fev. 2011. Academy and Industry Research Collaboration Center (AIRCC). http://dx.doi.org/10.5121/ijcses.2011.2102.

JOHNSTON, Alan B.; BURNETT, Daniel C.. **WebRTC:** APIs and RTCWEB Protocols of the HTML5 Real-time Web. 3. ed. St. Louis: Digital Codex Llc, 2014. 274 p.

KAMIENSKI, Carlos *et al.* **Colaboração na Internet e a Tecnologia Peer-to-Peer**. In: XXV CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 25., 2005, São Leopoldo. JAI 2005 - Congresso da SBC. São Leopoldo: Sbc, 2005. p. 1407 - 1454. Disponível em: https://www.researchgate.net/publication/255653983_Colaboracao_na_Internet_e_a_Tecnologia_Peer-to-Peer. Acesso em: 01 abr. 2019.

KANAZAWA, Kiyoshi; TAKAMI, Kazumasa. Standby P2P Network to Substitute for Client-Server Network in Event of Interruption of Communicatio. **Tencon 2018 - 2018 Ieee Region 10 Conference**, [s.l.], p.931-936, out. 2018. IEEE. http://dx.doi.org/10.1109/tencon.2018.8650292.

KANG, Beomgu *et al.* Popularity-based partial caching management scheme for streaming multimedia on proxy servers over IP networks. **2009 Ieee International Conference On Network Infrastructure And Digital Content**, [s.l.], p.586-590, nov. 2009. IEEE. http://dx.doi.org/10.1109/icnidc.2009.5360947.

KAZUNORI, Ueda; NORIO, Kimura. Network information sharing system with peer-to-peer network applications. **2015 17th Asia-pacific Network Operations And Management Symposium (apnoms)**, [s.l.], p.534-537, ago. 2015. IEEE. http://dx.doi.org/10.1109/apnoms.2015.7275407.

LEE, Jooyoen; KIM, Junghak; KIM, Shinho. Enhanced distributed streaming system based on RTP/RTSP in resurgent ability. **Fourth Annual Acis International Conference On Computer And Information Science (icis'05)**, [s.l.], p.1-5, 2005. IEEE. http://dx.doi.org/10.1109/icis.2005.59.

LI, Gaohe *et al.* Development and Research Based on WebRTC Mobile Phone Video Communication. **2019 Ieee 3rd Information Technology, Networking, Electronic And Automation Control Conference (itnec)**, [s.l.], p.2487-2490, mar. 2019. IEEE. http://dx.doi.org/10.1109/itnec.2019.8729024.

LORETO, Salvatore; ROMANO, Simon Pietro. Real-Time Communications in the Web: Issues, Achievements, and Ongoing Standardization Efforts. **Ieee Internet Computing**, [s.l.], v. 16, n. 5, p.68-73, set. 2012. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/mic.2012.115.

LORETO, Salvatore; ROMANO, Simon Pietro. **Real-Time Communication with WebRTC:** Peer-to-Peer in the Browser. Sebastopol: O'reilly Media, Inc, 2014.

MAJID, Hairudin Abdul *et al.* **P2P AUDIO AND VIDEO CALLING APPLICATION USING WEBRTC**. Arpn Journal Of Engineering And Applied Sciences, p. 1766-1770. fev. 2016. Disponível em: https://www.researchgate.net/publication/298711298_P2P_audio_and_video_calling_applic ation_using_WebRTC. Acesso em: 26 out. 2018.

MILOJICIC, Dejan S. *et al.* **Peer-to-Peer Computing**. Santa Barbara: Hewlett Packard Company, 2002. 52 p. Disponível em: https://www.labs.hpe.com/techreports/2002/HPL-2002-57R1.pdf. Acesso em: 29 jun. 2019.

NURMINEN, Jukka K. *et al.* P2P media streaming with HTML5 and WebRTC. **2013 Ieee Conference On Computer Communications Workshops (infocom Wkshps)**, [s.l.], p.63-64, abr. 2013. IEEE. http://dx.doi.org/10.1109/infcomw.2013.6970739.

OJAMAA, Andres; DÜÜNA, Karl. Assessing the security of Node.js platform. **2012 International Conference For Internet Technology And Secured Transactions**, [s.l.], p.348-355, 10 dez. 2012. Disponível em: https://ieeexplore.ieee.org/abstract/document/6470829. Acesso em: 26 maio 2019.

PANDEY, Nileshkumar; BEIN, Doina. Web application for social networking using RTC. **2018 Ieee 8th Annual Computing And Communication Workshop And Conference (ccwc)**, [s.l.], p.336-340, jan. 2018. IEEE. http://dx.doi.org/10.1109/ccwc.2018.8301692.

PARK, Changhee; RYU, Sukyoung. Scalable and Precise Static Analysis of JavaScript Applications via Loop-Sensitivity. In: 29TH EUROPEAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING (ECOOP 2015), 29., 2015, Dagstuhl. **Leibniz International Proceedings in Informatics (LIPIcs).** Dagstuhl: Schloss Dagstuhl--leibniz-zentrum Fuer Informatik, 2015. p. 735 - 756. Disponível em: http://drops.dagstuhl.de/opus/volltexte/2015/5245/. Acesso em: 26 maio 2019.

PIERLEONI, Paola *et al.* An innovative WebRTC solution for e-Health services. **2016 Ieee 18th International Conference On E-health Networking, Applications And Services (healthcom)**, [s.l.], p.1-6, set. 2016. IEEE. http://dx.doi.org/10.1109/healthcom.2016.7749444.

RESCORLA, Eric. **Security Considerations for WebRTC.** 2019. Disponível em: https://tools.ietf.org/html/draft-ietf-rtcweb-security-11. Acesso em: 18 jun. 2019.

RIBEIRO, Daniela Costa; COSTA, Daniel Gouveia. Utilizando Transmissões Multimídia IP em Tempo Real como Alternativa aos Métodos Tradicionais de Filmagem “ao vivo” em Emissoras de TV: um Estudo de Caso. In: XXXII SEMINÁRIO INTEGRADO DE SOFTWARE E HARDWARE, 25., 2005, São Leopoldo. **XXV**

Congresso da Sociedade Brasileira de Computação e XIII Workshop sobre Informática na Escola - WIE. Anais. São Leopoldo: Sbc, 2005. p. 1588 - 1600. Disponível em: http://www.lbd.dcc.ufmg.br/colecoes/semish/2005/001.pdf. Acesso em: 14 maio 2019.

RIGHI, Rafael da Rosa. **P2P-Role: Uma Arquitetura de Controle de Acesso Baseada em Papéis para Sistemas Colaborativos Peer-to-Peer.** 2005. 131 f. Tese (Doutorado) - Curso de Ciência da Computação, Universidade Federal de Santa Catarina, Florianópolis,, 2005. Disponível em: https://repositorio.ufsc.br/bitstream/handle/123456789/102583/213674.pdf?sequence=1. Acesso em: 05 abr. 2019.

RINALDI, Lucas. **Sistema de Atendimento ao Consumidor Utilizando a Tecnologia WebRTC.** 2018. 244 f. TCC (Graduação) - Curso de Curso de Sistemas de Informação, Universidade Federal de Santa Catarina, Florianópolis, 2018. Disponível em: https://repositorio.ufsc.br/handle/123456789/192175. Acesso em: 18 jun. 2019.

ROCHA, Roberto Oliveira. **WebRTC - Evolução na Web.** 2014. 111 f. Dissertação (Mestrado) - Curso de Sistemas e Tecnologias de Informação Para As Organizações, Instituto Politécnico de Viseu, Viseu, 2014. Disponível em: http://repositorio.ipv.pt/handle/10400.19/2554. Acesso em: 18 jun. 2019.

RODRIGUES, Rodrigo; DRUSCHEL, Peter. Peer-to-peer systems. **Communications Of The Acm**, [s.l.], v. 53, n. 10, p.72-82, 1 out. 2010. Association for Computing Machinery (ACM). http://dx.doi.org/10.1145/1831407.1831427.

ROMANO, Simon Pietro; LORETO, Salvatore; DAVIDS, Carol. Real Time Communications in the Web: Current Achievements and Future Perspectives. **Ieee Communications Standards Magazine**, [s.l.], v. 1, n. 2, p.20-21, 2017. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/mcomstd.2017.7992923.

ROUSSOPOULOS, Mema *et al.* 2 P2P or Not 2 P2P? **Lecture Notes In Computer Science**, [s.l.], p.33-43, 2005. Springer Berlin Heidelberg. http://dx.doi.org/10.1007/978-3-540-30183-7_4.

SANTOS, Alexandre Kreismann dos. **Soluções open source para interoperabilidade entre sistemas de videoconferência e Webconferência.** 2017. 70 f. TCC (Graduação) - Curso de Ciência da Computação, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2017. Disponível em: https://www.lume.ufrgs.br/handle/10183/153289. Acesso em: 18 jun. 2019.

SERRHINI. HOME USERS SECURITY AND THE WEB BROWSER INBUILT SETTINGS, FRAMEWORK TO SETUP IT AUTOMATICALLY. **Journal Of Computer Science**, [s.l.], v. 9, n. 2, p.159-168, 1 fev. 2013. Science Publications. http://dx.doi.org/10.3844/jcssp.2013.159.168.

SILVA, Juliana Chaves Braz da. **COMUNICAÇÃO EM TEMPO REAL COM WEBSOCKET APLICADO AO DESENVOLVIMENTO DE UM PROTÓTIPO PARA PRESCRIÇÃO DE TREINO DE MUSCULAÇÃO PARA DISPOSITIVOS MÓVEIS ANDROID.** 2015. 109 f. TCC (Graduação) - Curso de Ciência da Computação, Universidade do Extremo Sul Catarinense, Criciúma, 2015.

SINGH, Simar Preet; PASSI, Anjali. Real Time Communication. **International Journal Of Recent Development In Engineering And Technology.** Jalandhar, p. 141-144. 03 mar. 2014. Disponível em: http://www.ijrdet.com/files/Volume2Issue3/IJRDET_0314_23.pdf. Acesso em: 12 maio 2019.

SIVABALAKRISHNAN, M.; MANJULA, D.. Analysis of decision feedback using RTCP for multimedia streaming over 3G. **2008 International Conference On Computer And Communication Engineering**, [s.l.], p.1023-1026, maio 2008. IEEE. http://dx.doi.org/10.1109/iccce.2008.4580763.

SOCKET.IO (Org.). **What Socket.IO is.** 2019. Disponível em: https://socket.io/docs/ . Acesso em: 06 nov. 2019.

SOORYAVANSHI, Praduman; UPGANLAWAR, Saket; BHOSLE, Anand. Implementation of node.js server on Raspberry pi to control a remote vehicle for defense use. **2017 International Conference On Intelligent Sustainable Systems (iciss)**, [s.l.], p.816-819, dez. 2017. IEEE. http://dx.doi.org/10.1109/iss1.2017.8389290.

SREDOJEV, Branislav; SAMARDZIJA, Dragan; POSARAC, Dragan. WebRTC technology overview and signaling solution design and implementation. **2015 38th International Convention On Information And Communication Technology, Electronics And Microelectronics (mipro)**, [s.l.], p.1006-1009, maio 2015. IEEE. http://dx.doi.org/10.1109/mipro.2015.7160422.

SRIDHARAN, Manu *et al.* Correlation Tracking for Points-To Analysis of JavaScript. **Ecoop 2012 – Object-oriented Programming**, [s.l.], p.435-458, 2012. Springer Berlin Heidelberg. http://dx.doi.org/10.1007/978-3-642-31057-7_20.

STRIBNY, Martin; SMUTNY, Pavel. Using HTML5 web interface for visualization and control system. **Proceedings Of The 14th International Carpathian Control Conference (iccc)**, [s.l.], p.1-4, maio 2013. IEEE. http://dx.doi.org/10.1109/carpathiancc.2013.6560570.

TILKOV, Stefan; VINOSKI, Steve. Node.js: Using JavaScript to Build High-Performance Network Programs. **Ieee Internet Computing**, [s.l.], v. 14, n. 6, p.80-83, nov. 2010. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/mic.2010.145.

VAN LAMSWEERDE, Axel. **Requirements Engineering:** From System Goals to UML Models to Software Specifications. Chichester: John Wiley & Sons, 2009.

VINEETH, Nandhini *et al.* **CONTENT DISTRIBUTION VIA P2P NETWORKS: A SURVEY**. International Journal Of Engineering Applied Sciences And Technology. Bangalore, p. 237- 240. mar. 2017. Disponível em: http://www.ijeast.com/papers/237-240,Tesma204,IJEAST.pdf. Acesso em: 28 out. 2018.

WALKERDINE, J.; MELVILLE, L.; SOMMERVILLE, I.. **Dependability properties of P2P architectures. Proceedings. Second International Conference On Peer-to-peer Computing**,, [s.l.], p.1-2, 2002. IEEE Comput. Soc.

WEBRTC (Org.). **Frequent Questions: Is the WebRTC project owned by Google or is it independent?**. 2018. Disponível em: https://webrtc.org/faq/#is-the-webrtc-project-owned-by-google-or-is-it-independent. Acesso em: 04 nov. 2018.

WEBRTC (Org.). **Architecture.** 2019. Disponível em: https://webrtc.org/architecture/. Acesso em: 15 jun. 2019.

WITHERSPOON, Gloria *et al.* Evolving the Tactical Edge: Delivering Unified Capabilities (UC) and Mobile Enterprise Connectivity to the Deployed User. **2014 Ieee Military Communications Conference**, [s.l.], p.1269-1274, out. 2014. IEEE. http://dx.doi.org/10.1109/milcom.2014.211.

XIONG, Li; LIU, Ling. PeerTrust: Supporting Reputation-Based Trust for Peer-to-Peer Electronic Communities. **Ieee Transactions On Knowledge And Data Engineering**, [s.l.], v. 16, n. 07, p.843-857, jul. 2004. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/tkde.2004.1318566.

YOON, Soojin; JUNG, Jonghun; KIM, Hwankuk. Attacks on Web browsers with HTML5. **2015 10th International Conference For Internet Technology And Secured Transactions (icitst)**, [s.l.], p.193-197, dez. 2015. IEEE. http://dx.doi.org/10.1109/icitst.2015.7412087.

ZEIDAN, Adham; LEHMANN, Armin; TRICK, Ulrich. WebRTC enabled multimedia conferencing and collaboration solution. **Wtc 2014; World Telecommunications Congress 2014**, [s.l.], p.1-6, 3 jun. 2014. Disponível em: https://ieeexplore.ieee.org/abstract/document/6840017. Acesso em: 09 jun. 2019.

ZENG, Peng. A design of layered P2P-based IPTV system. **2011 Ieee 3rd International Conference On Communication Software And Networks**, [s.l.], p.578-581, maio 2011. IEEE. http://dx.doi.org/10.1109/iccsn.2011.6014791.

ZHU, Bin; MIAO, Huaikou; CAI, Lizhi. Testing a Web Application Involving Web Browser Interaction. **2009 10th Acis International Conference On Software Engineering, Artificial Intelligences, Networking And Parallel/distributed Computing**, [s.l.], p.589-594, 2009. IEEE. http://dx.doi.org/10.1109/snpd.2009.59.

APÊNDICE A – ARTIGO

Protótipo de Aplicação para Comunicação em Tempo Real Peer-to-Peer entre Navegadores Utilizando Webrtc

Gilberto V. Silva, Maicon M. Domingos

Curso de Ciência da Computação – Universidade do Extremo Sul Catarinense (UNESC)
Criciúma – SC – Brasil

gilbertovieirasilva@hotmail.com, maiconmatiola@gmail.com

Abstract. *Today's real-time communication service is one of the most important applications of the internet, this communication method enables its users to share information such as text, audio and video with each other for personal, social, educational or business reasons. There are a wide variety of applications with different communication solutions, but only a small portion of these use the fully decentralized Peer-to-Peer (P2P) concept. As a solution to this problem, a prototype application for P2P real-time communication between browsers using WebRTC was developed. It aims to enable direct communication between two or more browsers using WebRTC features in conjunction with a Node.js based signaling server. The results achieved at the end of the work were satisfactory, as a functional prototype with the ability to transmit text, audio and video directly between browsers using WebRTC was obtained and can serve as a basis for development for future real-time communication solutions.*

Resumo. *Atualmente o serviço de comunicação em tempo real é uma das aplicações mais importantes da internet, esse método de comunicação possibilita aos seus usuários compartilhar uns com os outros, por razões pessoais, sociais, educacionais ou de negócios, informações como texto, áudio e vídeo. Há uma grande variedade de aplicações, com diferentes soluções de comunicação, mas, somente uma pequena parcela destes usam o conceito totalmente descentralizado baseado em Peer-to-Peer (P2P). Como solução para este problema, foi desenvolvido um protótipo de aplicação para comunicação em tempo real P2P entre navegadores utilizando WebRTC. O mesmo tem como objetivo permitir a comunicação direta entre dois ou mais navegadores utilizando as funcionalidades do WebRTC em conjunto com um servidor de sinalização baseado em Node.js. Os resultados alcançados no final do trabalho foram satisfatórios, uma vez que se obteve um protótipo funcional com a capacidade de transmitir texto, áudio e vídeo diretamente entre navegadores utilizando o WebRTC e que pode servir como base para desenvolvimento para futuras soluções na área da comunicação em tempo real.*

1. Introdução

Atualmente o serviço de comunicação em tempo real é uma das aplicações mais importantes da internet, esse método de comunicação possibilita aos seus usuários compartilhar uns com os outros, por razões pessoais, sociais, educacionais ou de negócios, informações como texto, áudio e vídeo (BARRY; TOM, 2009, tradução nossa). Há uma grande variedade de aplicações, com diferentes soluções de comunicação, mas, somente uma pequena parcela destes usam o conceito totalmente descentralizado do Peer-to-Peer (P2P) (EDÄNGE, 2015, tradução nossa).

Segundo Majid et al. (2016, tradução nossa), no mercado atual existem inúmeras aplicações para comunicação em tempo real, como por exemplo: Skype, Google Hangouts, Face Time, sendo que a grande maioria dessas aplicações utilizam a arquitetura cliente/servidor, neste modelo de aplicação os usuários precisam utilizar um agente como um smartphone, estações de trabalho e outras aplicações de hardware e software, para que posteriormente o agente se conecte a um servidor central. Ainda de acordo com o autor, usar a arquitetura cliente/servidor nesse tipo aplicações pode aumentar muito o custo do sistema, devido a serviços como configuração e manutenção.

O WebRTC é um projeto de código aberto suportado pelo Google, Mozilla e Opera (WEBRTC, 2018). O objetivo do projeto é ampliar o modelo de navegação web, possibilitando aos navegadores comunicação direta de áudio, vídeo e dados em tempo real utilizando a arquitetura P2P (LORETO; ROMANO, 2014, tradução nossa). Ainda segundo os autores, o World Wide Web Consortium (W3C) junto com a Internet Engineering Task Force (IETF) são responsáveis por definir para o projeto WebRTC as Application Programming Interfaces (API) de JavaScript, as tags HTML5 padrão, e os protocolos de comunicação, com objetivo de possibilitar a qualquer aplicação web com WebRTC, funcionar em qualquer dispositivo, dispondo acesso seguro aos periféricos de entrada (webcams e microfones), para troca de mídia em tempo real P2P.

Com base nos fatores descritos anteriormente e considerando a necessidade de se implementar um protótipo de aplicação para comunicação em tempo real P2P entre navegadores, a API WebRTC mostrou ser uma alternativa bastante interessante, uma vez que é um projeto aberto e gratuito que permite desenvolver aplicações para comunicação direta em tempo real de texto, áudio, vídeo para navegadores, além disso, a iniciativa WebRTC é um projeto apoiado pelo Google, Mozilla e Opera, possibilitando que em futuro próximo o WebRTC possa ganhar ainda mais força.

3. Metodologia

Como linguagem de programação, foi definido o uso do JavaScript, devido principalmente ao fato de que os componentes WebRTC são acessados por meio de APIs JavaScript e também ao uso da plataforma Node.js na criação do servidor de sinalização. Isso torna possível o uso da mesma linguagem de programação tanto do lado do cliente como do servidor facilitado em parte o desenvolvimento do protótipo.

Para o desenvolvimento do servidor de sinalização, foi optado pelo uso da plataforma Node.js, que devido as suas características, as aplicações baseadas sobre a mesma possuem vantagens como o uso da linguagem JavaScript, alta escalabilidade e disponibilidade a um grande conjunto de APIs, com destaque a API do Socket.io.

A API do Socket.io em especial é de extrema importância para a funcionalidade do servidor de sinalização, uma vez que a mesma é responsável principalmente por desempenhar a função de transmissão e direcionamento de mensagens de sinalização, além do gerenciamento das salas de conversação e dos seus respectivos usuários.

No front-end foi definida a utilização do HTML5, uma vez que o mesmo disponibiliza recursos como as *tags* de áudio e vídeo que em conjunto com uma série de atributos particulares a cada uma delas, permitem a execução das mídias capturadas pelo WebRTC.

Para auxiliar o HTML5 na construção do front-end, foi optado pela utilização do *framework Bootstrap 4*, sendo que o mesmo possui um grande número de classes e estilos em

CSS em conjunto com algumas funcionalidades em JavaScript, disponibilizando ao usuário uma interface mais amigável e capaz de se adaptar a diversos tipos de dispositivos.

A arquitetura do protótipo desenvolvido respeita o modelo de triângulo proposto pelo WebRTC. Este modelo de arquitetura diferentemente do modelo de trapézio, é composto por um único servidor de sinalização, e o mesmo se conecta aos seus clientes por meio de conexões do tipo HTTP/WebSocket.

O servidor de sinalização, como o nome sugere tem o papel de sinalizar a presença de clientes que estejam dispostos a estabelecer uma conexão P2P com os demais participantes. Além disso o mesmo fica responsável pelas negociações de oferta, resposta e candidatos, transmitindo durante este processo, os parâmetros necessários para a criação de uma conexão direta entre os clientes envolvidos.

As aplicações Web do lado do cliente, ao se conectarem com o servidor de sinalização, em primeiro lugar comunicam aos demais clientes sobre sua presença, e em seguida, caso haja outros pares disponíveis para uma possível comunicação P2P as mesmas disponibilizam ao servidor de sinalização suas credenciais e seus protocolos de comunicação.

No caso de sucesso nas negociações entre as aplicações dos clientes, as mesmas criam uma conexão direta entre elas, utilizando a API PeerConnection presente no WebRTC, e a partir deste momento são capazes de gerenciar canais de comunicação possibilitando a transmissão de faixas de áudio, vídeo e texto.

4. Resultados Obtidos

Durante as pesquisas foram encontrados alguns trabalhos que tinham o WebRTC como objeto de estudo, porém em nenhuma das implementações foi encontrada a utilização do WebRTC na forma de multiconexões, sendo que o mesmo foi projetado para a comunicação par a par e não multipares.

A implementação do WebRTC para comunicação multipares, causou algumas dificuldades durante o processo de desenvolvimento, em um primeiro momento pela inexperiência do desenvolvedor em relação a tecnologia, e posteriormente pelo fato de a lógica utilizada para a criação de multiconexões não possuir nenhuma referência previa.

Durante os testes com o protótipo pode se encontrar um ponto fraco em relação ao modo de multiconexões inspirado em redes mesh, especialmente durante as transmissões multimídia, principalmente pelo fato de que, o número de conexões crescia proporcionalmente em relação ao número de usuários, exigindo muito da largura de banda da conexão. Porém como solução para este problema, foi decidido por limitar o número de integrantes à cinco por sala, resultando no máximo de quatro conexões por usuário.

Após o processo de desenvolvimento e testes, obteve-se um protótipo funcional, com o poder de gerenciar o uso de entradas multimídia, como microfones e webcans diretamente pelo navegador. Além disso o protótipo é capaz transmitir em tempo real, texto, áudio e vídeo em tempo real, diretamente entre dois ou mais navegadores, não importando onde os mesmos estejam.

5. Conclusão

Durante o desenvolvimento deste trabalho, ficou evidente a importância de serem desenvolvidas novas tecnologias para a comunicação em tempo real, uma vez que as aplicações desse gênero, atualmente, estão baseadas na arquitetura cliente/servidor e são disponibilizadas

por grandes empresas, assim as mesmas estão sujeitas a possíveis falhas, cobranças e até vazamento de informações.

Para o entendimento e desenvolvimento do protótipo proposto, foi necessário um levantamento bibliográfico bastante sólido, partindo desde os conceitos da arquitetura P2P e RTC, até o estudo de tecnologias para desenvolvimento web, e sobre o projeto WebRTC, construindo assim a base de conhecimento necessária para a concepção do protótipo.

A metodologia proposta para o desenvolvimento do protótipo, foi baseada em assuntos estudados anteriormente, como, engenharia de software, analise de requisitos e desenvolvimento web. A mesma em conjunto com o levantamento bibliográfico, foram responsáveis por permitir que tanto o objetivo geral quanto os objetivos específicos fossem alcançados, resultando assim, em um protótipo de aplicação para comunicação em tempo real P2P entre navegadores.

Devido à natureza do projeto WebRTC, durante o processo de implementação do protótipo, foi necessário o desenvolvimento de uma lógica de programação capaz de fazer comunicação entre vários pares, inspirada na tipologia de redes *mesh*, uma vez que o WebRTC que foi desenvolvido para fazer comunicação par a par e não multipares.

Durante os testes, a comunicação entre vários pares demonstrou um ponto fraco sendo este, o fato de que o número de conexões aumentar proporcionalmente ao número de usuários, causando um consumo cada vez maior da largura de banda disponível, principalmente durante as comunicações por stream de vídeo. Sendo assim foi necessária a limitação para no máximo de cinco, o número de integrantes para cada sala de bate-papo.

No entanto, podemos concluir que o trabalho desenvolvido pode atender todas as expectativas, alcançando os objetivos propostos no início do projeto, a partir da obtenção de um protótipo de aplicação para comunicação em tempo real entre navegadores utilizando WebRTC.

Referências

Barry, Bazara i. a.; Tom, Fatma m.. Instant Messaging: Standards, Protocols, Applications, and Research Directions. in: kutais, b.g.. internet policies and issues, volume 7. a: nova science publishers, inc, 2009. cap. 7. p. 1-14. disponível em: https://www.researchgate.net/publication/280307922_instant_messaging_standards_protocols_applications_and_research_directions. acesso em: 28 out. 2018.

Edänge, Simon. An Implementation and Performance Evaluation of a Peer-to-Peer Chat System. 2015. 40 f. Tese (Doutorado) - Curso de Computer Science, Faculty Of Computing Blekinge Institute Of Technology, Karlskrona, 2015. Disponível em:https://pdfs.semanticscholar.org/390c/633afd8b2ed1908fcb18a75ea13069f8dc3c.pdf. Acesso em: 26 out. 2018.

Loreto, Salvatore; Romano, Simon Pietro. Real-Time Communication with WebRTC: Peer-to-Peer in the Browser. Sebastopol: O'reilly Media, Inc, 2014.

Majid, Hairudin abdul et al. P2P Audio and Video Calling Application Using Webrtc. arpn journal of engineering and applied sciences, p. 1766-1770. fev. 2016. Disponível em: https://www.researchgate.net/publication/298711298_p2p_audio_and_video_calling_applic ation_using_webrtc. acesso em: 26 out. 2018. 449-460.

WEBRTC (Org.). Frequent Questions: Is the WebRTC project owned by Google or is it independent?. 2018. Disponível em: https://webrtc.org/faq/#is-the-webrtc-project-owned-by-google-or-is-it-independent. Acesso em: 04 nov. 2018.

.